Sigurnosni sustav za prepoznavanje ljudi u radnom prostoru robota zasnovan na AdaBoost algoritmu i stroju s potpornim vektorima

Šimundić, Valentin

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

SIGURNOSNI SUSTAV ZA PREPOZNAVANJE LJUDI U RADNOM PROSTORU ROBOTA ZASNOVAN NA ADABOOST ALGORITMU I STROJU S POTPORNIM VEKTORIMA

Diplomski rad

Valentin Šimundić

Osijek, 2021.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 12.07.2021.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime studenta:	Valentin Šimundić
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1094R, 06.10.2019.
OIB studenta:	61400622808
Mentor:	Prof.dr.sc. Robert Cupec
Sumentor:	Dr. sc. Petra Đurović
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Izv. prof. dr. sc. Emmanuel-Karlo Nyarko
Član Povjerenstva 1:	Prof.dr.sc. Robert Cupec
Član Povjerenstva 2:	Izv. prof. dr. sc. . Damir Filko
Naslov diplomskog rada:	Sigurnosni sustav za prepoznavanje ljudi u radnom prostoru robota zasnovan na AdaBoost algoritmu i stroju s potpornim vektorima
Znanstvena grana rada:	**Automatizacija i robotika (zn. polje elektrotehnika)**
Zadatak diplomskog rada:	Potrebno je implementirati metode za prepoznavanje i praćenje ljudi zasnovane na AdaBoost algoritmu i stroju s potpornim vektorima u ROS-u. Metode je potrebno integrirati sa stvarnim robotom i RGB-D kamerom. Razvijeni sustav treba osigurati sigurno kretanje čovjeka u radnom okruženju robota. Tema rezervirana za: Valentin Šimundić Sumentor s FERIT-a: Petra Đurović
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	12.07.2021.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 19.07.2021.

Ime i prezime studenta:	Valentin Šimundić
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1094R, 06.10.2019.
Turnitin podudaranje [%]:	4

Ovom izjavom izjavljujem da je rad pod nazivom**: Sigurnosni sustav za prepoznavanje ljudi u radnom prostoru robota zasnovan na AdaBoost algoritmu i stroju s potpornim vektorima**

izrađen pod vodstvom mentora Prof.dr.sc. Robert Cupec

i sumentora Dr. sc. Petra Đurović

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Ubrzani rast i napredak industrijske automatizacije procesnih pogona i sustava posljednjih desetljeća omogućen je zahvaljujući velikoj primjeni računala. Ovo je dovelo do povećanja učinkovitosti, kvalitete i fleksibilnosti u proizvodnji. Također, uvođenje industrijskih robota u proizvodnju omogućilo je smanjenje potrebe za ljudskom snagom u poslovima koji se ponavljaju te u poslovima koji mogu biti potencijalno opasni za ljude. Ipak, ljudska prisutnost je i dalje potrebna u takvim postrojenjima te je za kolaborativni rad čovjeka i robota potrebno osigurati ljude, robote i njihovu okolinu. Zbog svoje jednostavnosti i učinkovitosti, česte sigurnosne mjere u ovakvim slučajevima predstavljaju sigurnosna svjetlosna zavjesa, sigurnosna rešetka i slično, no takve mjere ne daju dovoljno informacija o objektu ili osobi koja prolazi. Međutim, sigurnost ljudi i robota u industrijskom okruženju moguće je postići sustavom u kojem se upravlja robotom na temelju informacije o prisutnosti čovjeka u okolini robota. Ovakav sustav treba dovoljno velikom frekvencijom davati informaciju o čovjeku, a ta informacija treba biti precizna i točna. U ovom se radu obrađuje sigurnosni sustav za prepoznavanje ljudi u radnom prostoru robota razvijen za potrebe tvrtke Danieli Systec d.o.o. Sustav se temelji na AdaBoost algoritmu te algoritmu stroja potpornih vektora (engl. *Support Vector Machine*, SVM). Tvrtka ima radionice i prostore u kojima djeluje robot, a blizu kojeg se nalazi izlaz u slučaju opasnosti. Upravo ovi slučajevi opasnosti su motivacija za realizaciju spomenutog sustava. Ukoliko dođe do opasnosti u prostoriji i čovjek krene prema izlazu, mora se osigurati da robot stane. Također, ukoliko čovjek prolazi kroz radni prostor robota noseći nešto, sustav mora zaustaviti robota. Ovakav sustav se povezuje s robotom i RGB-D kamerom u veći sustav kojemu je cilj upravljati robotom na osnovu informacija pozicije, brzine i smjera kretanja čovjeka, dobivenih sustavom za prepoznavanje čovjeka.

Rad je strukturiran na sljedeći način. U poglavlju 2, dan je pregled područja prepoznavanja ljudi na slikama u boji i dubinskim slikama. Nadalje, u poglavlju 3, opisuju se korišteni algoritmi AdaBoost i SVM koji predstavljaju temelj sustava za prepoznavanje ljudi. Robotski sustav, odnosno komponente koje grade ovaj sigurnosni sustav, opisane su u poglavlju 4. Struktura i implementacija rješenja u ROS okviru opisana je u poglavlju 5, dok je eksperimentalna evaluacija, zajedno s rezultatima, opisana u poglavlju 6. Na kraju, rad se zaključuje u poglavlju 7.

2. PREGLED PODRUČJA PREPOZNAVANJA LJUDI KORISTEĆI SLIKE U BOJI I DUBINSKE SLIKE

U ovom poglavlju dan je pregled literature iz područja prepoznavanja ljudi na slikama u boji i dubinskim slikama. Pružen je uvid u postojeća rješenja u danom području, rješenja su analizirana te su navedeni problemi takvih metoda i način prikladnog primjenjivanja algoritama ili dijelova metoda za postavljeni problem prepoznavanja ljudi.

Wu i drugi u radu *Real-Time Human Detection Using Contour Cues* [1], [2] predlažu sustav prepoznavanja ljudi koji se bavi dvama problemima: prepoznavanjem ljudi u stvarnom vremenu te identificiranjem najbitnijih izvora informacija. Tvrde kako u postojećim metodama „usko grlo“ (engl. *bottleneck*) predstavljaju efikasnost i brzina izvođenja algoritama jer se većina vremena računalne obrade svodi na izvlačenje značajki iz slika.

U [1], [2], predložen je sustav prepoznavanja ljudi u kojemu je kontura čovjeka izvor ključnih informacija za prepoznavanje, a usporedba piksela sa susjednim pikselima predstavlja ključ izvlačenja informacija iz konture. Također, predstavljen je i CENTRIST (Census Transform Histogram) deskriptor za prepoznavanje objekata koji sažeto kodira informacije o „znakovima usporedbi“ susjednih piksela. Cenzus transformacija (engl. *Census Transform*), na kojoj se temelji CENTRIST, uspoređuje intenzitet piksela s 8 susjednih piksela, što se može vidjeti na slici 2.1. Ako je intenzitet središnjeg piksela veći od ili jednak susjednom pikselu, na odgovarajuću lokaciju se upisuje bit 1, a u suprotnom slučaju se zapisuje bit 0. Generiranih 8 bitova se zapisuju u vektor te se prebacuju iz broja prikazanog u bazi 2 u broj prikazan u bazi 10. Ovakav se postupak ponavlja za sve piksele, čime se dobiva CT (*Census Transform*) slika. CENTRIST deskriptor ovdje predstavlja histogram CT slike ili njenog dijela s 256 stupaca.

$$\begin{array}{|c|c|c|} \hline 32 & 64 & 96 \\ \hline 32 & \mathbf{64} & 96 \\ \hline 32 & 32 & 96 \\ \hline \end{array} \Rightarrow \begin{array}{c} 1\ 1\ 0 \\ 1 \quad 0 \\ 1\ 1\ 0 \end{array} \Rightarrow (11010110)_2 \Rightarrow \mathrm{CT} = 214$$

Slika 2.1: Kodiranje Cenzus transformacijom [1]

Korištenjem CENTRIST-a s nekim od linearnih klasifikatora za prepoznavanje objekata, moguće je dizajnirati vrlo efikasne algoritme u kojima je potreban konstantan broj operacija (*O(1)* kompleksnosti) za svaki piksel [1], [2].

C^4 predstavlja algoritam koji koristi ranije opisane algoritme i načine. C^4 ispituje sve moguće prozore za prepoznavanje (veličine $h \times w$) na ulaznoj slici te klasificira svaki prozor kao objekt od interesa ili kao pozadinu. To je omogućeno pomoću kaskadne strukture C^4 okvira. Najprije se koristi linearni klasifikator H_{lin} koji ima veliku brzinu, ali malu preciznost. Nadalje, drugi klasifikator koji se koristi jest SVM zajedno s CENTRIST deskriptorom izračunatim na izlazu H_{lin} klasifikatora. Zadnji korak kaskade C^4 okvira predstavlja uklanjanje redundantnih detekcija metodom potiskivanja ne-maksimuma (engl. *Non-maximum suppression,* NMS). Detektirani prozor se izbacuje ukoliko preklapanje s drugim prozorom iznosi više od 60%, a sigurno prepoznavanje čovjeka označava prozor prepoznavanja koji s referentnim prozorom prepoznavanja ima preklapanje više od 50% [2]. Dok je za prepoznavanje ljudi korištena RGB slika i C^4 okvir, dobivanje udaljenosti ljudi od kamere dobiveno je pomoću stereo sustava kamera. Praćenje ljudi i izbacivanje pogrešnih prepoznavanja omogućeno je pomoću čestičnog filtera [1]. Rezultate sustava C^4 moguće je vidjeti na slici 2.2.

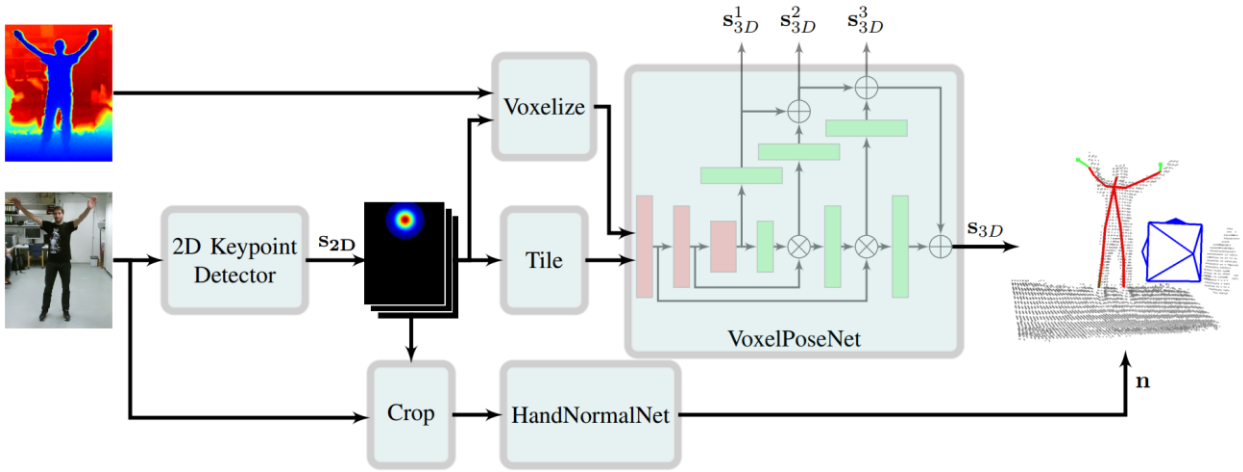
(a) C^4 (b) C^4+stereo (c) C^4+stereo, final

Slika 2.2: Rezultati C4 sustava za prepoznavanje i praćenje ljudi

U radu *3D Human Pose Estimation in RGB-D Images for Robotic Task Learning* [3], predstavljen je pristup za procjenu trodimenzionalne ljudske poze na temelju jedne RGB-D slike te detektora ljudskih ključnih točaka na toj slici. Predloženi sustav prvo procjenjuje ljudsku pozu na 2D slici boje. Nakon toga, neuronska mreža preuzima 2D pozu i dubinsku kartu na temelju kojih gradi punu 3D pozu čovjeka, a na kraju se procjenjuje i poza šake.

Ključne točke čovjeka u koordinatnom sustavu kamere dobivaju se na temelju slike boje, dubinske karte i njihove kalibracije. Za detekciju ključnih točaka na slici boje upotrijebljena je biblioteka

OpenPose koja kodira vjerojatnost pojedinih ključnih točaka čovjeka. Za dobivanje 3D procjene poze čovjeka, koristi se *VoxelPoseNet* neuronska mreža, čija je arhitektura temeljena na enkoder-dekoder neuronskoj mreži UNet. Procijenjene 2D točke se stavljaju jedna na drugu duž z-osi te se pomoću njih i dubinske karte stvara voxel[1] mreža središta čovjeka. Voxel mreža se zajedno s procijenjenim 2D točkama predaje *VoxelPoseNet* mreži kako bi se dobila puna 3D poza čovjeka. Procjena položaja ruke omogućena je korištenjem neuronske mreže *HandNormalNet* koja iz procijenjenih 2D točaka stvara vektore normale obje šake i pridružuje ih 3D pozi čovjeka [3]. Arhitekturu sustava za procjenu 3D ljudske poze opisane ranije moguće je vidjeti na slici 2.3.



Slika 2.3: Arhitektura sustava za procjenu 3D ljudske poze [3]

U radu *People Detection in RGB-D Data* [5], predstavljen je pristup za prepoznavanje ljudi iz RGB-D podataka u kojemu se koriste značajke HOG (engl. *Histogram of Oriented Gradients*) deskriptora na 2D slici kako bi se razvila metoda za prepoznavanje ljudi na dubinskoj slici zvana HOD (engl. *Histogram of Oriented Depths*).

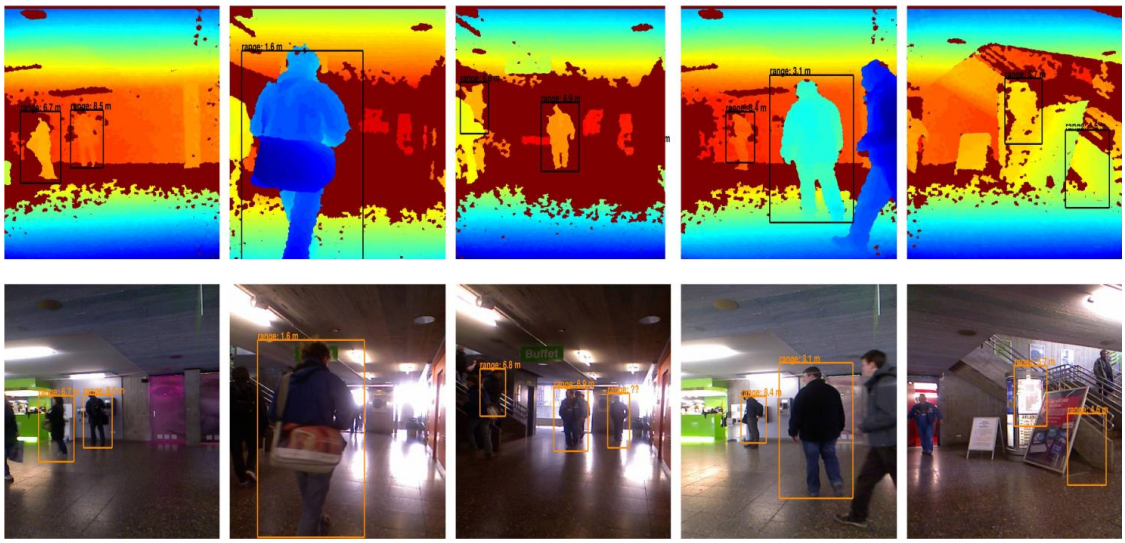
HOG deskriptor koristi prozor fiksne veličine podijeljen u gustu mrežu manjih ćelija. Za svaku ćeliju se izračunava gradijent orijentacije nad pripadajućim pikselima te se oni spremaju u vektor koji predstavlja histogram. Deskriptor koji okuplja sve histograme na slici koristi se za treniranje stroja s potpornim vektorima (engl. *Support Vector Machine, SVM*). HOG deskriptori mogu koristiti više veličina prozora te više pozicija, a sve ih je potrebno računati i trenirati sa SVM-om [5]. Detaljniji opisi HOG deskriptora i stroja s potpornim vektorima nalaze se u poglavlju 3.

[1] Voxel (engl. *Volume pixel*) predstavlja najmanju jedinicu 3D slike, a najčešće dolazi u obliku kocke. Predstavlja ekvivalent pikselu na 2D slici [4].

HOD algoritam koristi princip sličan principu rada HOG deskriptora na dubinskim slikama. Umjesto intenziteta boje pojedinih piksela, koristi se vrijednost dubine na pripadajućem pikselu pri pretvorbi orijentiranih gradijenata dubine u vektore histograma. Rezultat u obliku HOD značajki koristi se kao ulaz u linearni SVM. Budući da korišteni senzor za RGB-D slike, Microsoft Kinect v1, koristi 10 bitova za kodiranje dubine i nema ujednačenu rezoluciju kodiranja tih dubina (objektima koji se nalaze dalje od senzora pripada manje vrijednosti intervala kodiranja), napravljena je predobradba podataka. Predobradba podataka uključuje naglašavanje razdvajanja pozadine od prednjeg dijela slike kako bi se smanjile nelinearnosti u korištenim modelima [5].

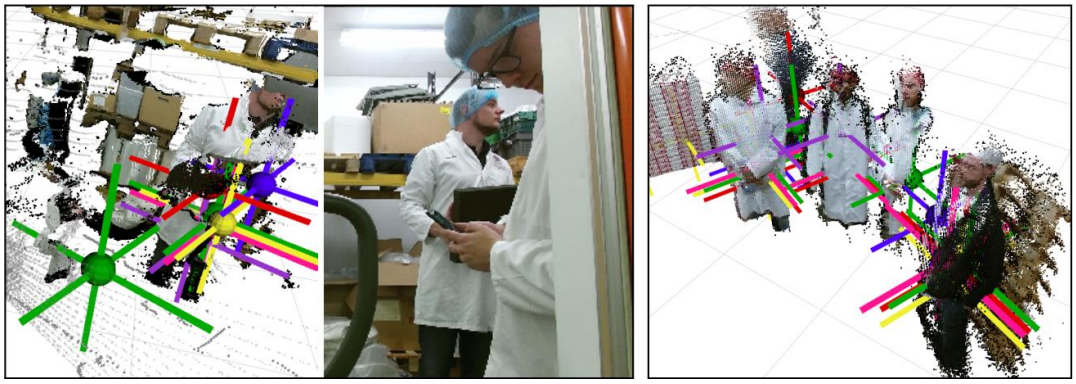
Opisani algoritmi koji uključuju HOG deskriptore i HOD značajke, kombiniraju se u jednu kaskadu naziva Combo-HOD. Combo-HOD uključuje iskorištavanje HOG deskriptora na RGB slikama te HOD značajki na dubinski slikama, pri čemu ih izračunava zasebno. Za svaki prozor prepoznavanja, kaskada zasebno računa HOG deskriptore i HOD značajke, a ukoliko ne postoji informacija dubine u spomenutom prozoru, kaskada se okreće samo HOG deskriptorima. Kako bi se mogla naći poveznica između dobivenih rezultata HOG deskriptora i HOD značajki, potrebna je kalibracija kamere. Spajanje rezultata dvaju algoritama omogućeno je skalarnim umnoškom dvaju deskriptora s hiperravninom dobivenom SVM-om [5].

Rezultati Combo-HOD kaskade mogu se vidjeti na slici 2.4. Metoda uspješno prepoznaje ljude na različitim udaljenostima i s različitim intenzitetom zaklanjanja. Lažni negativi (engl. *False negatives*) dobivaju se u slučajevima kada su podaci dobiveni s oba senzora izazovni, a lažni pozitivi (engl. *False positives*) dobiju se kada se zaklanjanje događa na oba senzora u isto vrijeme [5].



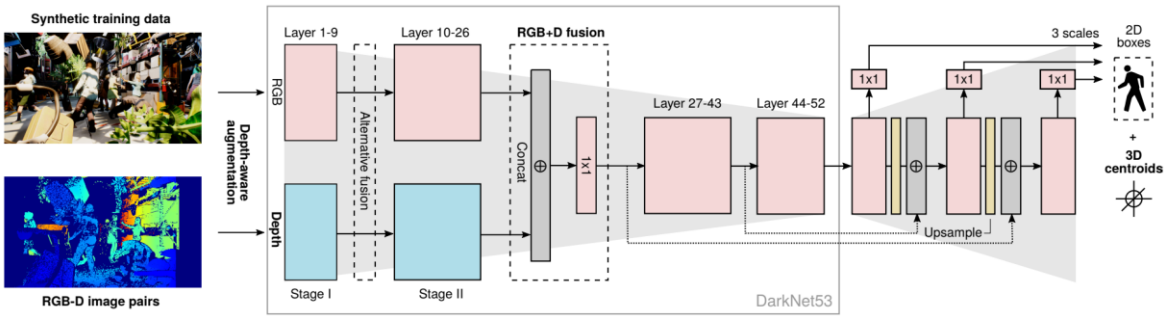
Slika 2.4: Prepoznavanje ljudi dobiveno Combo-HOD kaskadom [5]

U radu *Accurate detection and 3D localization of humans using a novel YOLO-based RGB-D fusion approach and synthetic training data* [6], predložen je pristup prepoznavanja ljudi koji proširuje mogućnosti YOLO v3 arhitekture na 3D te koristi značajke srednje razine (engl. *Mid-level features*) kako bi iskoristio informaciju iz oba izvora. Budući da su moguće situacije u kojima su ljudi na scenama zaklonjeni, predlažu se sljedeći pristupi. Prvi pristup iskorištava mogućnosti informiranijeg pristupa u kojemu koristi *offline* 3D procjenu ljudske poze kako bi se dobila referentna poza iz procijenjenih 3D zglobova tijela. Drugi pristup odabire fiksni, središnji dio tijela kao cilj regresije centroida čovjeka jer je stabilnije pričvršćen za ljudsko tijelo u slučajevima neobičnih poza tijela [6]. Primjer rada metode može se vidjeti na 2.5.



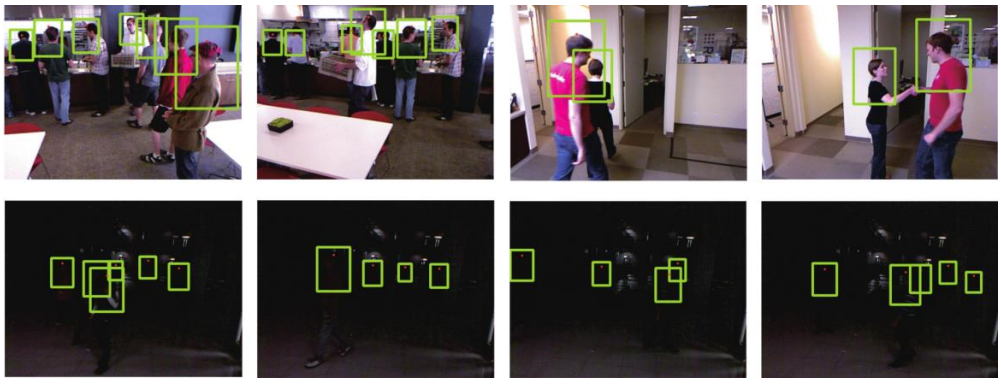
Slika 2.5: Metoda za 3D prepoznavanje ljudi korištenjem YOLO v3 neuronske mreže [6]

Budući da augmentacija ulaznih podataka neuronskih mreža može promijeniti skaliranje slike i time negativno utjecati na 3D percepciju objekta, u radu se predlaže augmentacija koja uključuje dubinu te održava originalnu skalu slike. Kako bi se omjer slike sačuvao pri treniranju mreže, uvedeno je ograničenje koje postavlja prozor čije dimenzije odgovaraju omjeru slike pri nasumičnom rezanju originalne slike. Također, uz poznatu matricu intrinzičnih parametara kamere **K**, provodi se skaliranje dubinskih veličina. [6] Neuronska mreža korištena u radu temelji se na YOLO v3 mreži. Prilagodba u odnosu na YOLO v3 odnosi se na prepoznavanje 3D centroida čovjeka. Korištena arhitektura YOLO v3 mreže je Darknet-53 koji omogućava uvođenje dodatnog kanala na ulazu za dubinske podatke. Nakon ulaznog sloja, podaci RGB slike i podaci dubinske slike se zasebno dupliciraju te se provodi fuzija iza slojeva 9 i 26. Fuzija podataka na sloju 26 spaja RGB i dubinske podatke u jednu cjelinu spajanjem te provođenjem 1×1 konvolucije. 1×1 konvolucije na kraju mreže kao rezultat daju koordinate centroida (c_x, c_y, c_z) čovjeka uz 2D prozor prepoznavanja [6]. Opisanu mrežu je moguće vidjeti na slici 2.6.



Slika 2.6: Arhitektura modificirane YOLO v3 mreže [6]

U radu *Robust 3D Human Detection in Complex Environments with Depth Camera* [7], predložen je dvostupanjski okvir prepoznavanja ljudi u složenim dinamičkim okruženjima koristeći dubinske podatke. U prvoj fazi koristi se princip lociranja *Head-top* kandidata (engl. *Candidate Head-top Locating*, CHL) kojim se izbjegava računska zahtjevnost principa pomičnih prozora (engl. *Sliding Window*) i smanjuje prostor za pretraživanje. CHL shema traži glavu čovjeka zbog njezine najmanje mogućnosti zaklanjanja. Svaki piksel na slici pripada određenom objektu ako ima manju Euklidsku udaljenost od zadane granice *t*, čime se na učinkovit način mogu pronaći kandidati (engl. *proposals*) čovjeka, a skup piksela s najvećom visinom na jednom kandidatu predstavlja prijedlog čovjekove glave [7]. Druga faza predstavlja kodiranje triju kanala informacije principom DMH (engl. *Depth Map, Multi-order depth template and Height difference map*) čija je svrha izvući geometrijske karakteristike kodirane na dubinskoj karti te se kao takvo koristi kao ulaz neuronskoj mreži. Neuronska mreža ima AlexNet arhitekturu koja je modificirana na način da, uz informaciju boje, koristi i informaciju dubine (DMH). Rad metode može se vidjeti na slici 2.7.



Slika 2.7: Primjer rada metode u različitim uvjetima [7]

3. TEORIJSKA PODLOGA

3.1. AdaBoost algoritam

Algoritmi podizanja (engl. *boosting algorithms*) predstavljaju skupinu algoritama koji pretvaraju slabe klasifikatore (engl. *weak learners*) u jake klasifikatore (engl. *strong learners*). Pripadaju skupini ansambl (engl. *ensemble*) metoda čiji je cilj stvoriti više modela te ih kombinirati kako bi se dobili bolji rezultati [8]. To se postiže na način da se za slabe klasifikatore koriste algoritmi s velikom varijancom i pogreškom (s malo boljim uspjehom klasifikacije od nasumičnim pogađanjem), poput stabala odluke dubine 1, perceptrona, naivnog Bayesovog klasifikatora itd. Ovi algoritmi se treniraju iterativno te svakom algoritmu pripadne težina (engl. *weight*), ovisno o uočenim slabostima u klasifikaciji. Nakon što svi slabi klasifikatori izbace svoj rezultat, uzima se težinski prosjek rezultata koji predstavlja konačni rezultat jakog klasifikatora. Postoji nekoliko algoritama podizanja, a najpoznatiji od njih su: algoritam adaptivnog podizanja (engl. *Adaptive Boosting*, AdaBoost), podizanje gradijenta (engl. *Gradient Boosting*, GBM), LightGBM, CatBoost i drugi [9].

Nakon objavljivanja koncepta algoritma podizanja, Freund i Schapire su 1995. godine predstavili AdaBoost algoritam, nadogradnju na originalnu ideju algoritma podizanja kojom su riješili određene probleme ranije predloženih algoritama [10].

Neka postoji skup podataka:

$$x_i \in \mathbb{R}^n, y_i \in \{-1, 1\}, \tag{3-1}$$

gdje *x* predstavlja ulazne podatke, *n* predstavlja dimenziju ulaznih podataka, a *y* predstavlja označene izlazne podatke u obliku brojeva 1 ili -1. AdaBoost postavlja težine svim skupovima u trening skupu podataka kako bi se označio njihov značaj u cjelokupnom skupu podataka. Ako je težina veća, podatak ima veći značaj i obrnuto. U početnom trenutku, svi podaci imaju istu vrijednost težine *w* koju je moguće dobiti sljedećim izrazom:

$$w = \frac{1}{N} \in [0, 1], \tag{3-2}$$

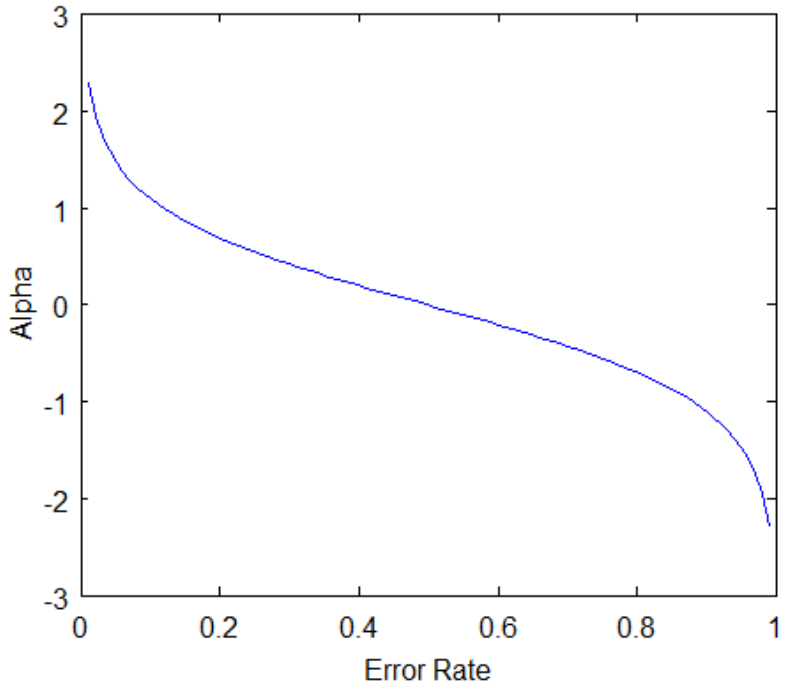
gdje *N* označava ukupan broj podataka u skupu ulaznih podataka. Zbroj težina uvijek će biti 1, a težina svakog podatka ima vrijednost između 0 i 1. Osim podataka, svaki klasifikator ima svoju težinu:

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right), \tag{3-3}$$

gdje α_t predstavlja težinu slabog klasifikatora t, a ϵ_t predstavlja broj pogrešnih klasifikacija u trening skupu podataka podijeljenu s brojem podataka u tom skupu (stopa pogrešnih klasifikacija, engl. *Misclassification rate*). Stopa pogrešnih klasifikacija može se dati izrazom [11]:

$$\epsilon_t = \frac{\sum_{i:\, h_t(x_i)\neq y_i} D_t(i)}{N} \tag{3-4}$$

gdje D_t predstavlja vektor težina ulaznih podataka, i predstavlja pripadajuću ulaznu veličinu, a $h_t(x_i)$ predstavlja izlaz slabog klasifikatora za ulazni podatak x_i. U obzir se uzimaju samo oni izlazni podaci koji nisu jednaki pripadajućim referentnim podacima. Slika 3.1 prikazuje graf ovisnosti težine slabog klasifikatora i stope pogrešaka. Moguće je vidjeti kako manja stopa pogrešaka dovodi do veće težine klasifikatora i obrnuto. U slučaju kada stopa pogreška iznosi 0.5, odnosno klasifikator izbacuje rezultate na temelju nasumičnog pogađanja, pridružuje mu se težina 0 te se ignorira u krajnjoj odluci. Ukoliko je vrijednost stope pogreške veća od 0.5, vrijednost težine je tada manja od 0. To znači da se za dani klasifikator uzima rezultat suprotan rezultatu koji taj klasifikator izbaci.



Slika 3.1: Graf ovisnosti težine klasifikatora i stope pogrešaka [12]

Ažuriranje težina ulaznih veličina potrebno je obaviti na sljedeći način:

$$D_{t+1}(i) = \frac{D_t(i)e^{-\alpha_t y_i h_t(x_i)}}{Z_t} \tag{3-5}$$

gdje Z_t predstavlja faktor normalizacije. Budući da je D_t distribucija, normaliziranjem težina je potrebno dobiti da je zbroj svih težina jednak 1 (varijabla Z_t). y_i predstavlja referentni podatak *i*-te ulazne veličine, a $h_t(x_i)$ izlaz klasifikatora t za ulazni podatak x_i. D_{t+1} predstavlja novu normaliziranu distribuciju ulaznih veličina za slabi klasifikator h_{t+1} u kojemu su povećane težine koje je prethodni klasifikator h_t pogrešno klasificirao. Time algoritam naglašava „teže“ slučajeve i povećava robusnost.

Na kraju, konačni jaki klasifikator ima jednadžbu:

$$H(x) = sgn\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right) \tag{3-6}$$

gdje T predstavlja broj slabih klasifikatora, $h_t(x)$ predstavlja izlaz slabog klasifikatora za ulazni podatak *x*. U ovom slučaju se na izlazu koristi funkcija *sgn*, odnosno signum funkcija koja određuje predznak dobivenog broja budući da su izlazi ograničeni na -1 i 1. U jednadžbi (3-6) moguće je vidjeti kako je krajnji rezultat samo zbroj svih slabih klasifikatora zajedno s njihovim težinama [13]–[15].

3.2. Stroj s potpornim vektorima

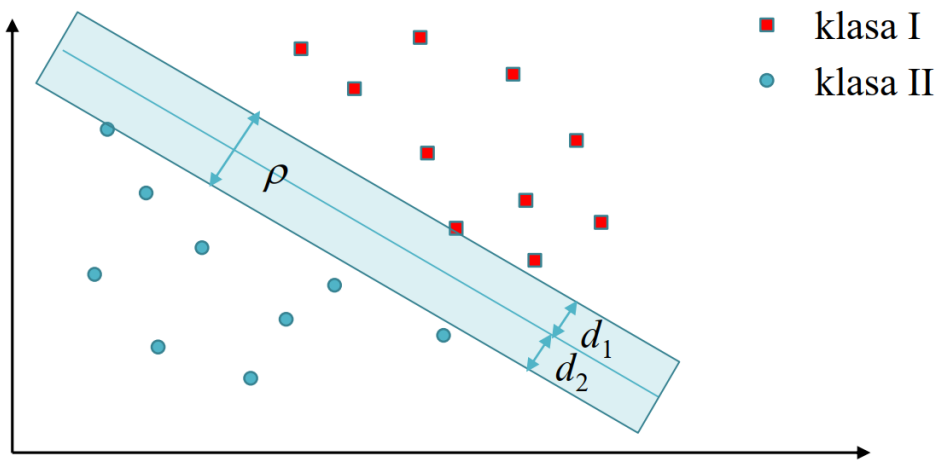
Stroj s potpornim vektorima (engl. *Support Vector Machine*, SVM) predstavio je Vladimir Vapnik 1995. godine u istraživačkom centru AT&T Bell Laboratories. Algoritam se danas uspješno koristi u mnogim područjima, poput kategorizacije teksta, prepoznavanja govora, informacijske sigurnosti i sl. [16] Zbog svoje podloge u statističkoj teoriji učenja i razvijanja u okviru istraživačkog centra, algoritam je od početka bio usmjeren na praktičnu primjenu. Statistička teorija učenja ili Vapnik-Chervonenkis teorija predstavlja karakteristike svojstava strojnog učenja koji omogućuju dobro generaliziranje na neviđene podatke. Iako je SVM najprije korišten za zadatke optičkog prepoznavanja znakova (engl. *Optical character recognition*, OCR), ubrzo se počeo koristiti u svrhu prepoznavanja objekata i regresije [17], [18].

Iako se SVM može koristiti i za regresiju, ovdje će se opisati SVM za klasifikaciju podataka. Algoritam je moguće koristiti za linearnu i nelinearnu klasifikaciju, a optimizacijski cilj

predstavlja maksimizacija margine razdvajanja (engl. *margin*). Margina razdvajanja predstavlja udaljenost između razdvajajuće hiperravnine i uzoraka trening podataka koji su najbliži toj hiperravnini. Spomenuti uzorci podataka nazivaju se potporni vektori (engl. *support vectors*) [19]. SVM vrši klasifikaciju uzoraka u dvije klase konstrukcijom spomenute hiperravnine u visoko-dimenzionalnom prostoru. Hiperravnina može se opisati sljedećim izrazom [20]:

$$\boldsymbol{w}^T\boldsymbol{x} + b = 0, \tag{3-7}$$

gdje **w** predstavlja normalu ravnine, **x** predstavlja ulazni vektor, a *b* predstavlja pomak od ishodišta koordinatnog sustava. U dvodimenzionalnom prostoru, hiperravnina predstavlja pravac, a u trodimenzionalnom prostoru predstavlja ravninu. Slika 3.2 prikazuje optimalni pravac koji razdvaja dvije klase dobiven algoritmom SVM. Margina razdvajanja označena je s ρ, a oznake d_1 i d_2 predstavljaju udaljenosti pravca do najbližeg uzorka klase I, odnosno klase II.



Slika 3.2: Pravac koji razdvaja dvije klase dobiven pomoću SVM algoritma [20]

Svi ulazni podaci moraju zadovoljavati jedan od dva uvjeta [20], [21]:

$$\boldsymbol{w}_0^T\boldsymbol{x}_i + b_0 \geq 1, za\ y_i = 1, \tag{3-8}$$

$$\boldsymbol{w}_0^T\boldsymbol{x}_i + b_0 \leq -1, za\ y_i = -1, \tag{3-9}$$

gdje y_i predstavlja klasu pripadajućeg ulaznog podatka i te, budući da se radi o binarnoj klasifikaciji, može biti iznosa 1 ili –1, a $\boldsymbol{w}_0$ i b_0 predstavljaju parametre stroja s potpornim vektorima. Kombinacijom izraza (3-8) i (3-9), dobije se sljedeći izraz [20], [21]:

$$y_i(\boldsymbol{w}_0^T \boldsymbol{x}_i + b_0) - 1 \geq 0, \forall i, \tag{3-10}$$

a međusobno paralelne ravnine R_1 i R_2 se mogu opisati izrazima [20], [21]:

$$\boldsymbol{w}_0^T \boldsymbol{x}_i + b_0 = 1 \; za \; R_1, \tag{3-11}$$

$$\boldsymbol{w}_0^T \boldsymbol{x}_i + b_0 = -1 \; za \; R_2. \tag{3-12}$$

Budući da su d_1 i d_2 udaljenosti hiperravnine od R_1 i R_2, ekvidistantnost hiperravnine daje sljedeći izraz [20]:

$$\frac{\rho}{2} = d_1 = d_2 = \frac{1}{\|\boldsymbol{w}_0\|}, \tag{3-13}$$

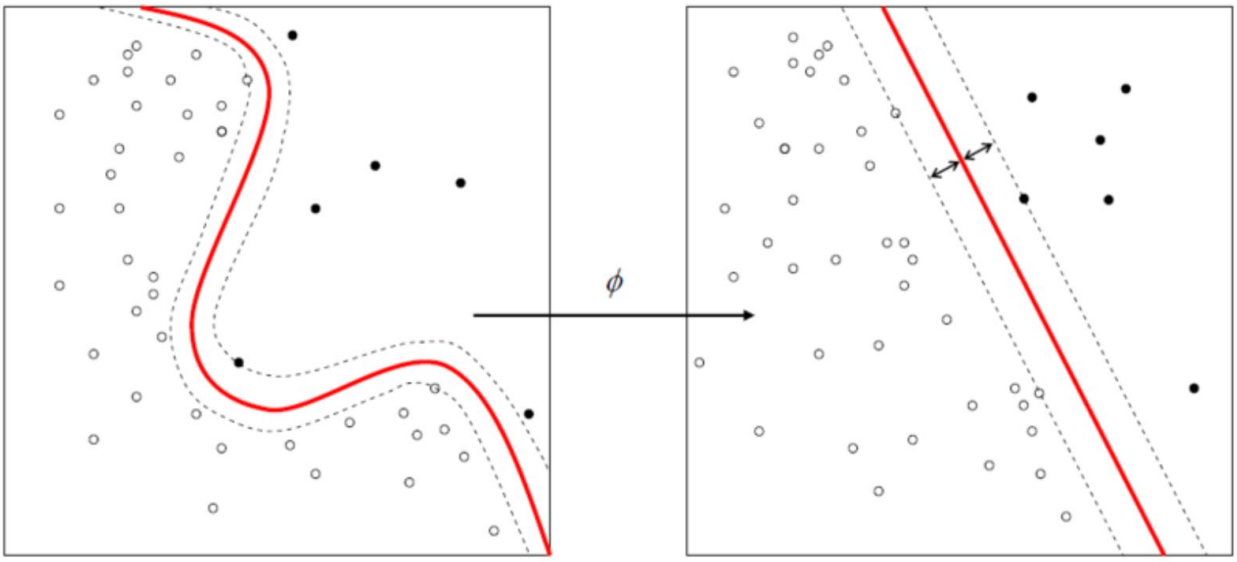
iz čega slijedi da je izraz za širinu margine stroja s potpornim vektorima sljedeći:

$$\rho = \frac{2}{\|\boldsymbol{w}_0\|}. \tag{3-14}$$

Iz jednadžbe (3-14) moguće je zaključiti kako se minimiziranjem euklidske norme $\boldsymbol{w}_0$ postiže maksimizacija širine margine ρ. Drugim riječima, potrebno je pronaći minimum sljedeće funkcije:

$$\Phi(\boldsymbol{w}) = \frac{1}{2}\boldsymbol{w}^T \boldsymbol{w}, \tag{3-15}$$

takav da vrijedi izraz (3-10). Optimalnu hiperravninu, odnosno minimum funkcije dane izrazom (3-15), moguće je dobiti korištenje optimizacije pomoću Lagrangeovih multiplikatora. Za linearno neodvojive razrede, nelinearnim preslikavanjem ulaznog prostora u novi prostor značajki više dimenzionalnosti, raste vjerojatnost linearne odvojivosti, o čemu govori Coverov teorem (Slika 3.3) [20].



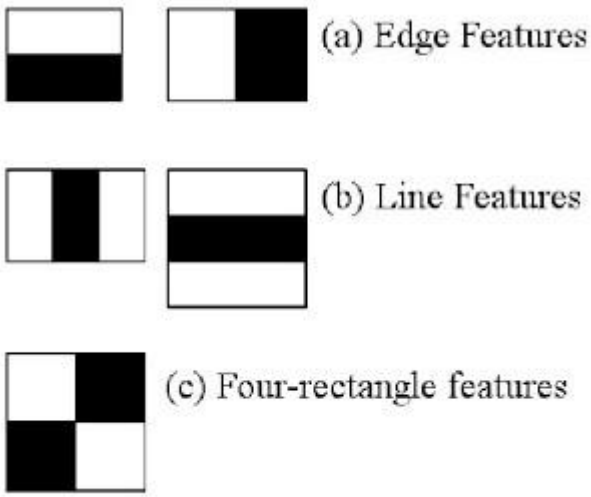
Slika 3.3: Preslikavanje ulaznog prostora (lijevo) u novi prostor više dimenzionalnosti [20]

3.3. Haar značajke

Detekciju objekata koristeći klasifikatore temeljene na Haar značajkama, 2001. godine predstavili su Paul Viola i Michael Jones u radu „*Rapid Object Detection using a Boosted Cascade of Simple Features*“ [22]. Algoritam je još nazvan Viola-Jones algoritam te, iako ga je moguće koristiti i u druge svrhe, najčešće se koristi za prepoznavanje lica.

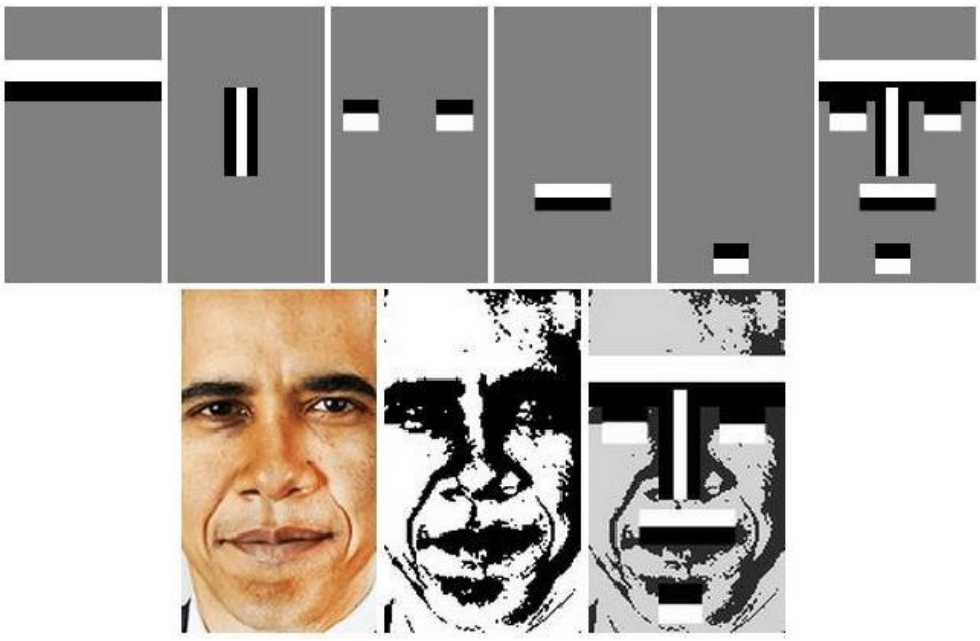
Najprije, algoritam pretvara sliku u sliku sivih tonova (engl. *grayscale*) kako bi se olakšalo izračunavanje Haar značajki. Haar značajke predstavljaju pravokutne okvire promjenjivih veličina koji se koriste za usporedbu međusobnog odnosa piksela [23]. Postoje tri tipa Haar značajki (Slika 3.4):

- Značajke rubova (engl. *Edge features*)
- Značajke linija (engl. *Line features*)
- Značajke četiri četverokuta (engl. *Four-rectangle features*)



Slika 3.4: Haar značajke [22]

Značajke rubova služe za prepoznavanje rubova na slici. Najčešće se koriste za izlučivanje značajki iz očiju, obrva i čela. Značajke linija prepoznaju linije na slici poput nosa ili za praćenje kretanja očiju. Značajke četiri četverokuta koriste se najčešće za traženje dijagonalnih linija i određenih značajki koje se ističu na licu. Ovisno o svjetlu, moguće je pronaći bradu, čeljust, bore i sl. Ove značajke najčešće nisu pretežito bitne u prepoznavanju lica zbog njihovog velikog broja koji bi bio uzrok sporom algoritmu. Također, budući da traži značajke koje se ističu na licu, algoritam bi prepoznavao samo određena lica [23]. Primjena svih značajki se može vidjeti na slici 3.5.



Slika 3.5: Primjena Haar značajki na licu [23]

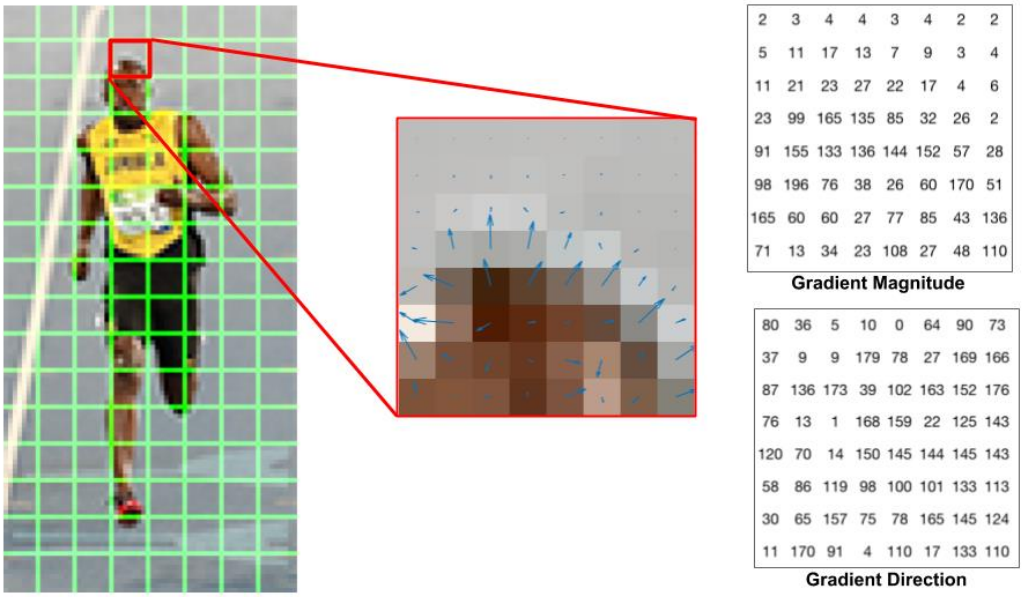
Za izračun i pronalazak značajki se koriste sve moguće pozicije i veličine pravokutnih okvira. Za svaku značajku je potrebno izračunati zbroj piksela ispod bijelih i crnih pravokutnika okvira. Budući da na slici može biti velik broj značajki, moguće je imati i velik broj nebitnih značajki. Kako bi se izbjegla velika računalna zahtjevnost algoritma, Haar značajke se često koriste u kombinaciji s AdaBoost algoritmom [22].

3.4. HOG deskriptor

Deskriptor značajki predstavlja pojednostavljenu sliku ili dio slike izvlačenjem korisnih i izbacivanjem suvišnih informacija slike [24]. Postoji više deskriptora, a najpoznatiji su: HOG (engl. *Histogram of Oriented Gradients*), SIFT (engl. *Scale Invariant Feature Transform*) i SURF (engl. *Speeded-Up Robust Feature*) [25].

Histogram orijentiranih gradijenata (engl. *Histogram of Oriented Gradients,* HOG) predstavlja deskriptor značajki koji se koristi za izvlačenje značajki iz slika, a najčešće se koristi u područjima računalnog vida i prepoznavanja objekata. Koncept HOG-a opisao je Robert K. McConnell 1986. godine, a u širu praktičnu primjenu je ušao 2005. godine kada su ga istraživači INRIA-e[2] iskoristili za prepoznavanje pješaka [26], [27]. Glavna karakteristika HOG deskriptora jest traženje gradijenta orijentacije u lokaliziranim područjima slike. Za svako pojedino područje slike, generira se histogram koristeći spomenute gradijente i orijentacije vrijednosti piksela. Slika 3.6 prikazuje primjer dijeljenja slike na ćelije u kojima se na pojedinim pikselima izračunavaju magnituda i smjer gradijenta.

[2] National Institute for Research in Computer Science and Automation



Slika 3.6: Primjer korištenja HOG deskriptora na slici [24]

HOG deskriptor se računa na dijelu slike veličine 64×128, a omjer slike treba biti 1:2. Računanje gradijenta se odvija u *x* i *y* smjerovima pomoću sljedećih filtera (engl. *kernels*):

$$g_x = [-1 \quad 0 \quad 1], \tag{3-16}$$

$$g_y = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \tag{3-17}$$

gdje g_x i g_y predstavljaju filtere u *x*, odnosno *y* smjeru. Magnituda i smjer gradijenta piksela izračunavaju pomoću sljedećih izraza:

$$g = \sqrt{g_x^2 + g_y^2}, \tag{3-18}$$

$$\theta = \arctan\left(\frac{g_y}{g_x}\right) \tag{3-19}$$

gdje g predstavlja magnitudu gradijenta, a θ smjer gradijenta piksela. Nadalje, slika se dijeli na ćelije veličine 8×8 piksela. Stvara se histogram s 9 stupaca (engl. *bins*) koji odgovaraju kutovima od 0 do 160° s korakom 20 [24]. Za svaki se interval računa suma iznosa gradijenata čija orijentacija pripada danom intervalu. Dobiveni histogram se normalizira tako da se stvara ćelija veličine 16×16 piksela te prolazi slikom pomoću metode pomičnog prozora [28]. Blok 16×16 sadrži četiri prethodno izračunata histograma koje je moguće spojiti u jedan vektor veličine 36×1

te normalizirati L2 (Euklidskom) normom [24]. Ovakav histogram je moguće koristiti sa strojevima s potpornim vektorima za detekciju ljudi i objekata s dobrim rezultatima [29].

3.5. Metoda za prepoznavanje i praćenje ljudi

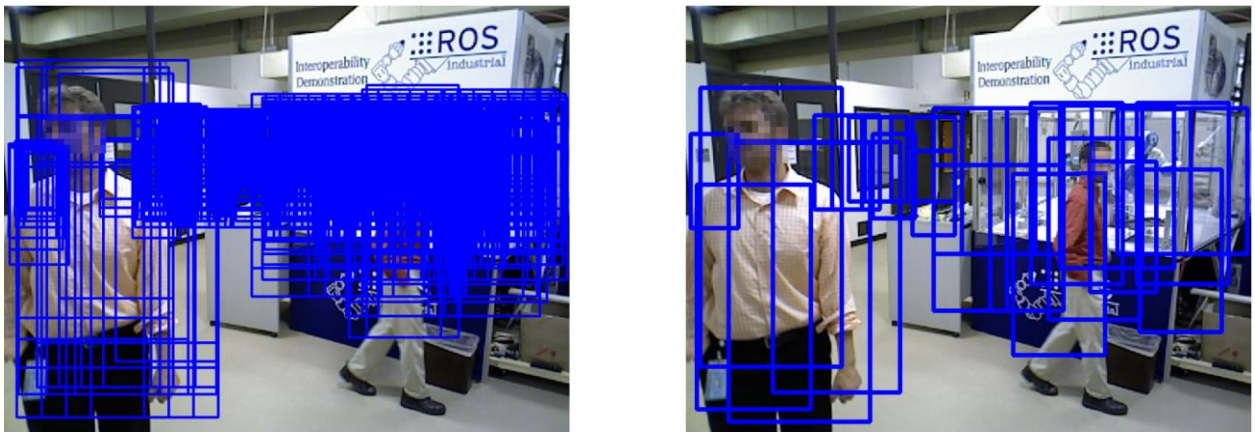
Rad „RGB-D Human Detection and Tracking for Industrial Environments“ [30] bavi se problemom prepoznavanja i praćenja ljudi u industrijskim okruženjima te je projekt otvorenog kôda[3]. U radu se predlažu algoritmi za prepoznavanje i praćenje ljudi koji koriste podatke o boji i dubini na slici za praćenje ljudi u stvarnom vremenu. Praćenjem kretanja ljudi, omogućava se procjenjivanje njihove buduće pozicije i budućeg smjera kretanja. S tim informacijama je moguće izbjegavanje sudaranja robota s čovjekom i smanjivanje vremena potrebnog za obavljanje posla. Sustavi u kojima ljudi i roboti rade zajedno u industrijskim okruženjima imaju nekoliko ograničenja koje je potrebno zadovoljiti kako bi se osigurala sigurnost čovjeka, robota te njegovog okruženja. Potrebno je prepoznati sve ljude u okruženju koje se promatra, bez obzira na zaklonjenost, sustav mora dovoljno velikom frekvencijom izbacivati rezultate (od 15 do 30 Hz [30]), a kašnjenje bi trebalo držati ispod 0.2 s [30] bez žrtvovanja performansi prepoznavanja ili praćenja. Algoritmi koji se koriste u ovoj metodi su prilagođeni za rad u ROS-u te je za treniranje i testiranje algoritama snimljen skup podataka „ROS-Industrial People Dataset“ koji se sastoji od 11000 RGB slika i pripadnih slika dispariteta u prostoru koji imitira industrijsko okruženje, no skup podataka nije dostupan javnosti. Tri tipa algoritama koji se koriste u ovoj metodi uključuju: algoritam konzistentnosti (engl. *Consistency algorithm*), algoritmi temeljeni na boji i algoritmi temeljeni na dubini [30].

3.5.1. Algoritam konzistentnosti

Postoje dva ograničenja koja provjeravaju je li prozor prepoznavanja u skladu s čovjekom na tlu. Prvo ograničenje predstavlja visinu na slici u odnosu na disparitet. To znači da čovjek određene visine na slici u pikselima mora imati disparitet koji ga stavlja na razumnu udaljenost. Odnosno, nije moguće da je prozor prepoznavanja nekoliko piksela, a čovjek se nalazi blizu kamere. Drugo ograničenje prisiljava ljude, odnosno prozore prepoznavanja, da se nalaze na ravnini koja predstavlja tlo.

[3] https://github.com/ros-industrial/human_tracker/tree/develop

Također, prozori, čije se površine preklapaju više od 80 % s ostalim prozorima, se uklanjaju (Slika 3.7). Ovo ograničenje smanjuje broj prepoznavanja koja se trebaju analizirati, modul se naziva CROW (engl. *Consistency with Removal of Overlapping Windows*) [30].



Slika 3.7: Algoritam konzistentnosti - lijevo: bez uklanjanja prozora; desno: s uklanjanjem prozora [30]

3.5.2. Algoritmi temeljeni na bojama i dubini

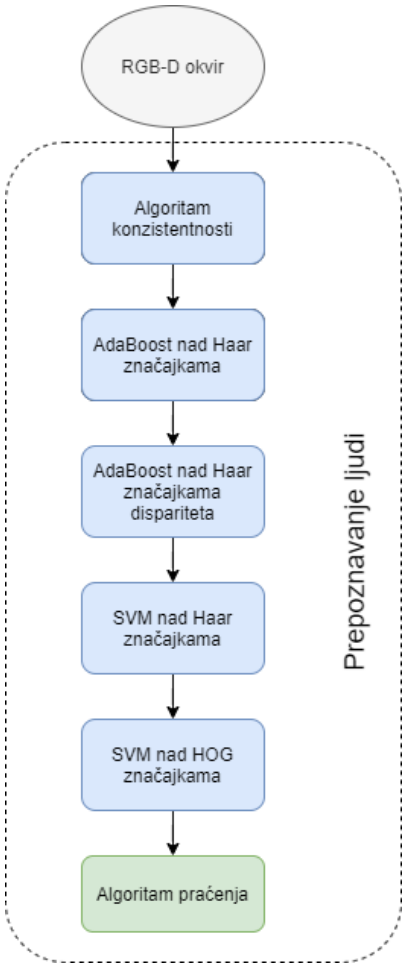
U ovom dijelu se koriste Haar značajke s algoritmima AdaBoost i SVM te HOG deskriptori sa SVM algoritmom. Haar značajke predstavljaju ulazne podatke za slabe klasifikatore AdaBoost algoritma. Algoritam je treniran s težinama koje teže ispravnim detekcijama u odnosu na ograničavanje lažnih pozitiva kako bi se eliminirali prozori za koje je sigurno da nisu ljudi. Haar značajke također su korištene i kao ulazi u stroj s potpornim vektorima u kojima su dobiveni rezultati jednaki originalnom radu, ali uz brže izvođenje. Kako bi se izvukle samo potrebne informacije o ljudima te izbacili gradijenti pozadine, korišteni su HOG deskriptori zajedno sa strojevima s potpornim vektorima. Ova metoda izračunava HOG deskriptore samo u određenim dijelovima prozora prepoznavanja, označujući samo siluetu čovjeka [30].

Iako metoda omogućava korištenje više algoritama dubine, u ovom diplomskom radu koristit će se Haar značajke dobivene sa slike dispariteta zajedno s AdaBoost algoritmom. Ključna razlika u implementaciji, u odnosu na isti algoritam temeljen na slikama boje, je da potraga za značajkama u obzir uzima i piksele s nepoznatim disparitetom. Pikseli s nepoznatim disparitetom imaju tada vrijednost 0 [30].

3.5.3. Algoritam praćenja

Praćenje ljudi u ovom radu koristi rezultate niza ranije opisanih algoritama kako bi se stvorila poveznica između postojećih praćenja i novih prepoznavanja. Algoritam se bazira na izračunavanju histograma boje na RGB slikama vođenih čestičnim filterom u *x-y* ravnini te koristi

računalne niti (engl. *threads*) kako bi se ljudi prepoznali čak i kada niz algoritama ne da zadovoljavajući rezultat. Broj praćenja koji je moguće u istom trenutku izvršavati ograničen je višenitnim (engl. *multithreading*) sposobnostima računala [30].



Slika 3.8: Korišteni niz algoritama (izradio autor prema slici u [30])

Unatoč drugim algoritmima koje je moguće koristiti u metodi, najbolje rezultate daje niz algoritama prikazan na slici 3.8 gdje se koriste ranije opisani postupci i algoritmi. Svi algoritmi implementirani su u C++ programskom jeziku te koriste mogućnosti ROS-a i biblioteka otvorenog kôda za 2D i 3D računalni vid.

4. ROBOTSKI SUSTAV

4.1. ROS

Robot Operating System (ROS) predstavlja razvojni okvir za pisanje programske podrške robota. ROS predstavlja zbirku alata, biblioteka i pravila kojima je cilj pojednostaviti zadatke stvaranja složenog sustava ponašanja robota na širokom spektru platformi za robote [31]. Projekt ROS započeo je Morgan Quigley 2007. godine pod imenom *Switchyard* kao dio STAIR[4] projekta, dok se glavnina razvoja ROS-a odvijala u tehnološko-istraživačkom laboratoriju *Willow Garage* [32].

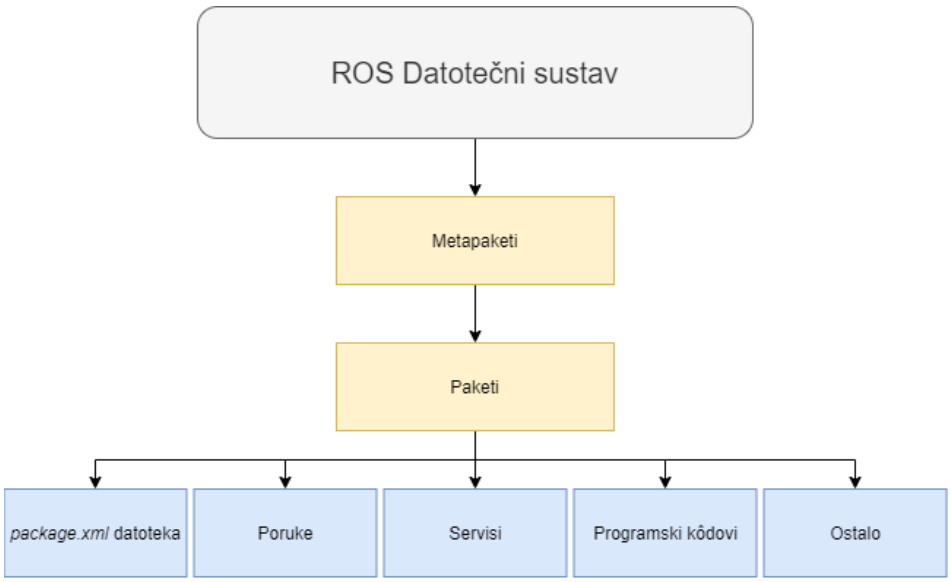
Filozofija ROS-a prati filozofiju UNIX-a u određenim točkama što omogućava programerima naviknutih na UNIX lakšu i prirodniju prilagodbu. ROS sustavi se u suštini sastoje od većeg broja manjih povezanih računalnih programa koji konstantno izmjenjuju poruke (engl. *messages*). Ove poruke putuju iz jednog programa u drugi bez ikakvog centralnog usmjerivanja, što karakterizira *peer-to-peer* (P2P) sustave. Ovakav način izmjene informacija usložnjava pozadinu sustava, ali omogućava prijenos većeg broja podataka [33]. Temeljenost na alatima je još jedna od značajki ROS-a. Ovdje ne postoji nikakav univerzalni alat koji omogućava potpunu vizualizaciju, pretraživanje stabla kôda ili zapisivanje podataka, već se omogućuje razvijanje novih alata koji će određene poslove odrađivati zasebno. Također, omogućeno je i korištenje većeg broja programskih jezika za različite namjene, a neki od njih uključuju: C++, Python, Java-u, JavaScript, MATLAB i druge [33]. Konvencije ROS-a omogućavaju stvaranje zasebnih biblioteka koje je moguće „upakirati“ (engl. *wrap*) kako bi mogli ostvariti komunikaciju s drugim ROS modulima. Zasebni sloj koji omogućava zasebne biblioteke također omogućava i korištenje istih izvan opsega ROS okvira, pritom omogućavajući lakšu integraciju s drugim aplikacijama. Sama jezgra ROS-a izdana je pod BSD[5] licencom, što omogućava komercijalnu i nekomercijalnu upotrebu te iskorištava mogućnosti projekta otvorenog kôda [33].

4.1.1. Komponente ROS okvira

Osim što omogućava alate i biblioteke, ROS također ugrađuje mogućnosti koje su nalik funkcijama operacijskog sustava, poput apstrakcije sklopovlja (engl. *hardware abstraction*) ili

[4] engl. *Stanford AI Robot*
[5] engl. *Berkeley Software Distribution*

upravljanje paketima. Iz tog razloga ga se još naziva i meta-operacijskim sustavom [32]. Upravljanje datotekama u ROS-u se izvodi na način prikazan na slici 4.1.



Slika 4.1: Datotečni sustav ROS-a (izradio autor prema slici u [32])

Paketi predstavljaju najjednostavniju jedinicu programske podrške ROS-a. Sadrže jedan ili više ROS programa, biblioteka, konfiguracijskih datoteka i dr. koji predstavljaju jednu jedinicu. Datoteka *package.xml* sadrži informacije o paketu, autoru, licenci, ovisnosti o drugim paketima i dr. te je obavezna datoteka paketa [32]. Metapaketi predstavljaju skup slabo povezanih paketa koji ne sadrže programski kôd, već samo referencu na ostale pakete koje povezuju [34]. Poruke (*.msg* datoteke) predstavljaju tipove informacija koje se prenose iz jednog ROS procesa u drugi, a servisi omogućavaju komunikaciju tipa zahtjev/odgovor (engl. *request/reply*) između ROS procesa [32].

Projekti u ROS-u uglavnom se sastoje od sljedećih komponenata: glavni čvor, korisnički čvorovi, poslužitelj parametara, poruke, teme (engl. *topics*) i servisi. Glavni čvor omogućava registraciju svakog čvora prilikom njegovog pokretanja te komunikaciju između čvorova. Čvorovi su procesi koji odrađuju određeni zadatak ili dio zadatka. Pisani su pomoću korisničkih biblioteka, pomoću kojih se implementiraju razne funkcionalnosti poput komunikacije između čvorova. Koriste se kako bi se olakšao rad s kompleksnim sustavima razvijanjem manjih podsustava. Poslužitelj parametara dio je glavnog čvora i omogućava spremanje informacija u središnju lokaciju kojoj svaki čvor može pristupiti i koju može izmijeniti. Izmjena informacija čvorova izvodi se preko tema. Teme predstavljaju sabirnice poruka, a čvorovi mogu slati informacije preko teme objavljivanjem (engl. *Publishers*) ili primati informacije s teme (engl. *Subscribers*) [32].

4.2. ABB-IRB 2400L

ABB-IRB 2400L industrijski je robotski manipulator koji se najčešće koristi u raznim sustavima zavarivanja. Robot se koristi u elektrolučnom zavarivanju, automatiziranom zavarivanju plazmom i automatizaciji laserskog zavarivanja, ali ga je moguće koristiti i u zadacima hvatanja određenog predmeta i premještanja na drugo mjesto (engl. *Pick and Place*) [35]. Robotski manipulator prikazan je na slici 4.2.

Slika 4.2: Robotski manipulator ABB-IRB 2400L [36]

Koristi razne ABB-ove upravljače (engl. *controllers*) robota (poput S4C i S4CPlus [36]), uključujući i ABB IRC5 upravljač koji se koristi u ovom radu. Robotski manipulator predstavlja produženu verziju manipulatora ABB-IRB 2400 te nudi veći dohvat, ali manje maksimalno opterećenje. Programiranje samog upravljača manipulatora moguće je preko pripadajuće programske podrške koju je razvio ABB, a naziva se *RobotStudio.* Programski jezik visoke razine korišten za programiranje manipulatora naziva se RAPID. Specifikacije robota mogu se vidjeti u tablici 4.1, a sâm robotski manipulator se može vidjeti na slici 4.2.

Tablica 4.1: Tehničke specifikacije robotskog manipulatora ABB-IRB 2400L (izradio autor prema podacima u [37])

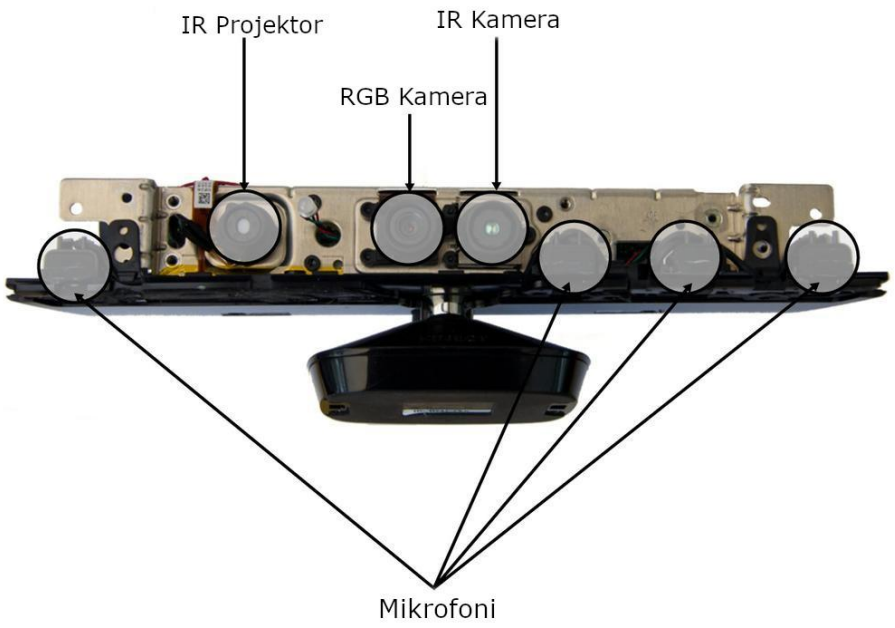
ABB-IRB 2400L	
Broj osi	6
Maksimalno opterećenje [kg]	7
Maksimalni doseg [mm]	1800
Ponovljivost [mm]	0.05
Upravljač	IRC5

4.3. Microsoft XBOX 360 Kinect

Microsoft XBOX 360 Kinect platforma (MS Kinect v1) predstavljena je 2010. godine kao dodatak igraćoj konzoli XBOX 360 za videoigre u kojima se upravlja ljudskim pokretom [38]. Unatoč glavnoj primjeni u videoigrama, Kinect je stekao i veliku primjenjivost u robotici kao jeftini 3D senzor.

Slika 4.3: Microsoft Kinect v1 [39]

Kinect omogućava snimanje RGB slike te dubinske slike i stvaranje 3D oblaka točaka (engl. *Point Cloud*). Senzor za dubinu sastoji se od infracrvenog projektora i infracrvene kamere te daje dubinsku sliku temeljenu na principu strukturirane svjetlosti (engl. *Structured light principle*) [39]. Slika 4.4 prikazuje raspored senzora Kinecta. Moguće je vidjeti kako su IR projektor i IR kamera razdvojene, što Kinectu omogućava provođenje postupka triangulacije za dobivanje dubinske slike. Maksimalna rezolucija dubinske i RGB kamera jest 640×480 piksela u 30 slika u sekundi na USB 2.0 sučelju.



Slika 4.4: Senzori Kinecta (izradio autor prema slici u [40])

Prednosti korištenja Kinecta u robotici uključuju nisku cijenu, visoku rezoluciju slike, manje odudarajućih vrijednosti (engl. *outliers*), visoka razina rekonstrukcije scene i dr. Ograničenja koja se pojavljuju pri korištenju dubinske kamere Kinecta uključuju [40]:

- poteškoće pri snimanju na dnevnom svjetlu,
- nemogućnost dobivanja dubine s prozirnih i reflektivnih površina te
- točnost opada s udaljenošću od kamere.

5. IMPLEMENTACIJA ALGORITAMA

U ovom poglavlju opisat će se implementacija algoritama prepoznavanja i praćenja ljudi opisanih u poglavlju 3.5 te implementacija algoritma za upravljanje robotskim manipulatorom.

5.1. Implementacija algoritama prepoznavanja i praćenja ljudi

Algoritmi za prepoznavanje i praćenje ljudi su u potpunosti osposobljeni za rad u ROS-u. Originalni algoritmi napisani su za ROS Fuerte (Ubuntu 12.04 LTS) te se kao način upravljanja datotekama koristio *rosbuild*. Ovakav način upravljanja datotekama je zamijenjen *catkin* sustavom koji se bazira na *CMake*-u. Budući da se u radu koristi ROS Melodic za Ubuntu 18.04 LTS, cijeli je sustav prebačen na *catkin* sustav prema uputama postavljenim na službenoj stranici ROS-a[6]. Također, budući da OpenCV 3 u vrijeme razvijanja originalnog projekta nije postojao, dijelovi projekta koji se tiču računalnog vida su napisani koristeći biblioteku OpenCV 2. ROS Melodic pri instalaciji dolazi s bibliotekom OpenCV 3 te je bila potrebna zamjena i ovog dijela sustava prema uputama OpenCV organizacije[7]. Sve promjene su uspješno testirane na ROS Melodic i ROS Kinetic distribucijama.

5.1.1. Struktura programskog rješenja

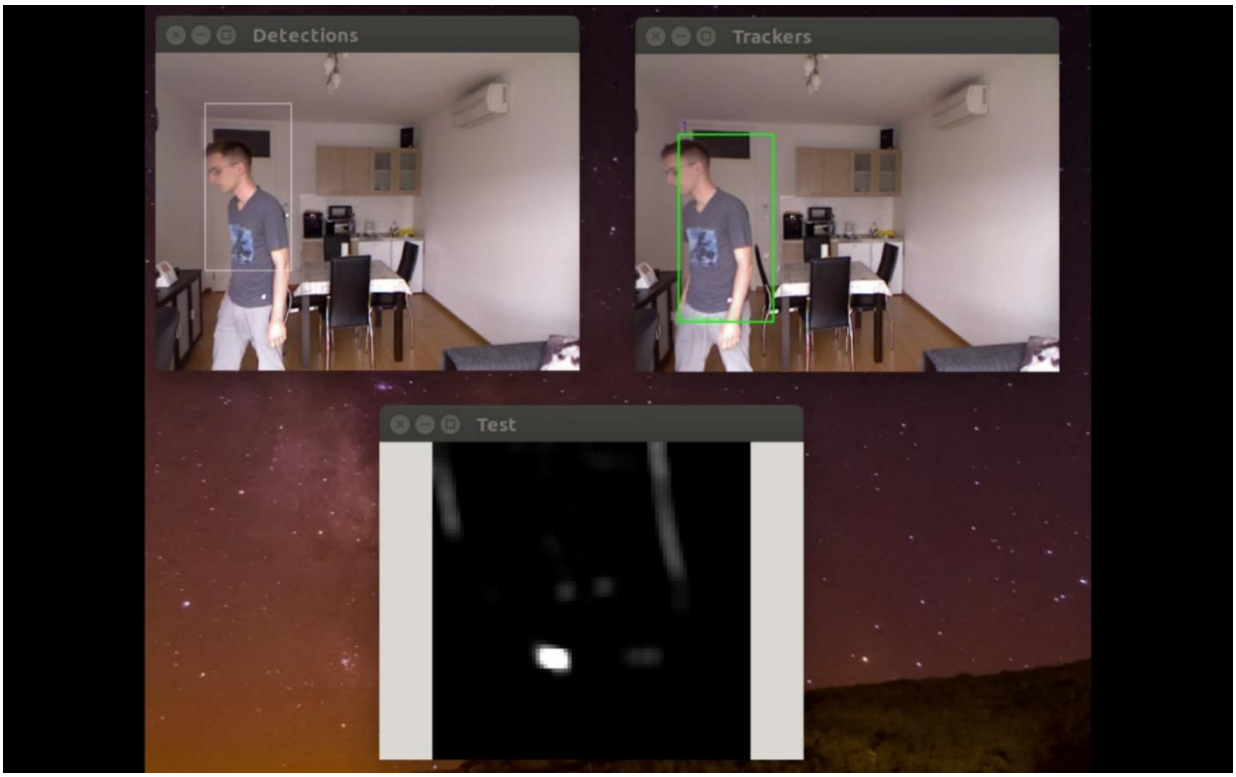
Cijeli projekt sastoji se od većeg broja paketa potrebnih za pravilan rad cijelog sustava. Ti paketi uključuju: *HaarAda*, *HaarDispAda*, *HaarSvm*, *HogSvm*, *consistency*, *object_tracking*, *roi_msgs*, *visualize_bbox*, *System_Launch* te *dataset_testing* pakete. Svaki paket izvršava određeni dio posla sustava, a razmjenom obrađenih informacija između paketa se postiže krajnji rezultat prepoznavanja – koordinate čovjeka u prostoru. Sav programski kôd pisan je u C++ programskom jeziku.

Svaki paket sadrži jedan pripadajući čvor (engl. *node*) u kojemu se poziva objekt klase pripadajućeg klasifikatora. Na taj način se u *HaarAda* paketu pokreće čvor *HaarAdaNode*, a u njemu objekt klase *HaarAdaClassifier* obavlja klasifikaciju ulaznih podataka. Također, čvorovi u kojima se obavlja glavni dio prepoznavanja ljudi (*HaarAda*, *HaarDispAda*, *HaarSvm*, *HogSvm* i *consistency*) omogućavaju više načina radova: prepoznavanje (engl. *detect*), prikupljanje podataka (engl. *accumulate*), treniranje (engl. *train*), evaluacija (engl. *evaluate*) i učitavanje (engl. *load*). Način rada učitavanja, iz pripadajuće datoteke u kojoj se nalaze težine klasifikatora, učitava

[6] http://wiki.ros.org/catkin/migrating_from_rosbuild
[7] https://docs.opencv.org/3.4/db/dfa/tutorial_transition_guide.html

spremljeni model te prebacuje način rada u prepoznavanje. U načinu rada prepoznavanja se pomoću inicijaliziranog i učitanog modela provodi prepoznavanje iz ulaznih podataka te se podaci dobiveni prepoznavanjem prosljeđuju sljedećem klasifikatoru u drugom čvoru. Prije treniranja, koristi se način rada prikupljanja podataka u kojemu se prikupljaju anotirane slike i pripadajući granični okviri (engl. *Bounding boxes*). Nakon toga se način rada prebacuje u treniranje u kojemu se, nad prikupljenim podacima, treniraju klasifikatori.

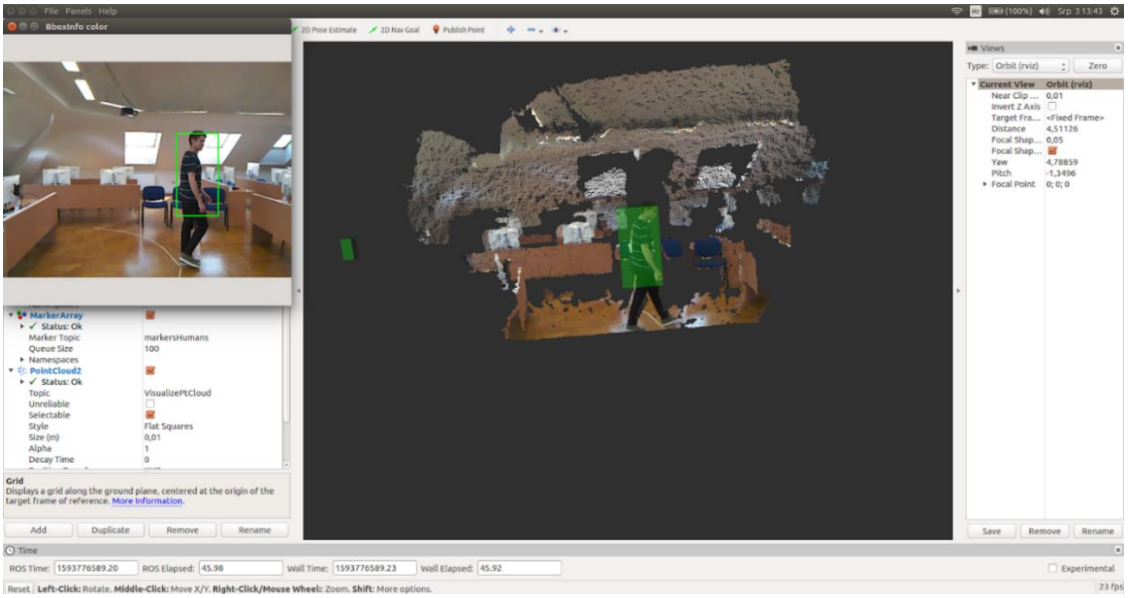
Praćenje ljudi nalazi se u zasebnom čvoru *ObjectTrackingNode* paketa *object_tracking*. Koristeći podatke o slici boje, slici dispariteta, graničnim okvirima prethodnih čvorova i parametrima kamere, čvor omogućava praćenje jednog čovjeka ili više ljudi. Maksimalni broj praćenja koje algoritam može izvršiti ovisi o snazi procesora, odnosno o njegovoj sposobnosti korištenja više niti (engl. *threads*) u isto vrijeme. Također, praćenje je omogućeno čak i u slučajevima kada čovjek izađe iz kadra kamere ili kada je zaklonjen. U takvim slučajevima sustav, pomoću fiksnog zadanog vremena istjecanja, čeka da to vrijeme istekne prije nego što makne određeno prepoznavanje. Ukoliko se u međuvremenu osoba opet pojavi u kadru i sustav ju prepozna, praćenje će pridijeliti osobi pripadajući ID, a vrijeme istjecanja se više ne uzima u obzir. Na kraju, čvor praćenja omogućava i prosljeđivanje informacije o čovjeku putem poruka i tema. Informacije o čovjeku se zapisuju u zasebno izrađene poruke koje se nalaze u paketu *roi_msgs* pod nazivom *HumanEntries.msg* i *HumanEntry.msg*. *HumanEntry* tip poruke sadrži sljedeće informacije o čovjeku: vrijeme prepoznavanja, ID pridijeljen osobi, centroid osobe (*x*, *y* i *z* koordinate), središte gornjeg brida graničnog okvira osobe (*x*, *y* i *z* koordinate), komponente brzine čovjeka, visinu i širinu graničnog okvira te kovarijancne podatke između *x*, *y* i *z* koordinata. Također, u ovom se čvoru omogućava i prikazivanje prepoznavanja i graničnih okvira u 2D. Rad metode može se vidjeti na slici 5.1. Gornji lijevi prozor na slici predstavlja trenutno prepoznavanje algoritama, drugi prozor prikazuje praćenje trenutne instance prepoznavanja, a treći prozor prikazuje rad čestičnog filtera korištenog za praćenje. Cijeli sustav se pokreće pomoću jedne *.launch* datoteke koja se nalazi u paketu *System_Launch.*



Slika 5.1: Rad algoritama za prepoznavanje i praćenje ljudi

5.1.2. Programsko rješenje 3D vizualizacije

U metodi je omogućena i jednostavna 3D vizualizacija koja omogućava prikazivanje graničnog okvira na čovjeku u 3D oblaku točaka postavljanjem kvadra na mjesto prepoznavanja. Vizualizacija se nalazi u paketu *visualize_bbox* te ju je u radu moguće isključiti kako bi se poboljšale performanse rada metode. 3D vizualizacija se može vidjeti na slici 5.2.



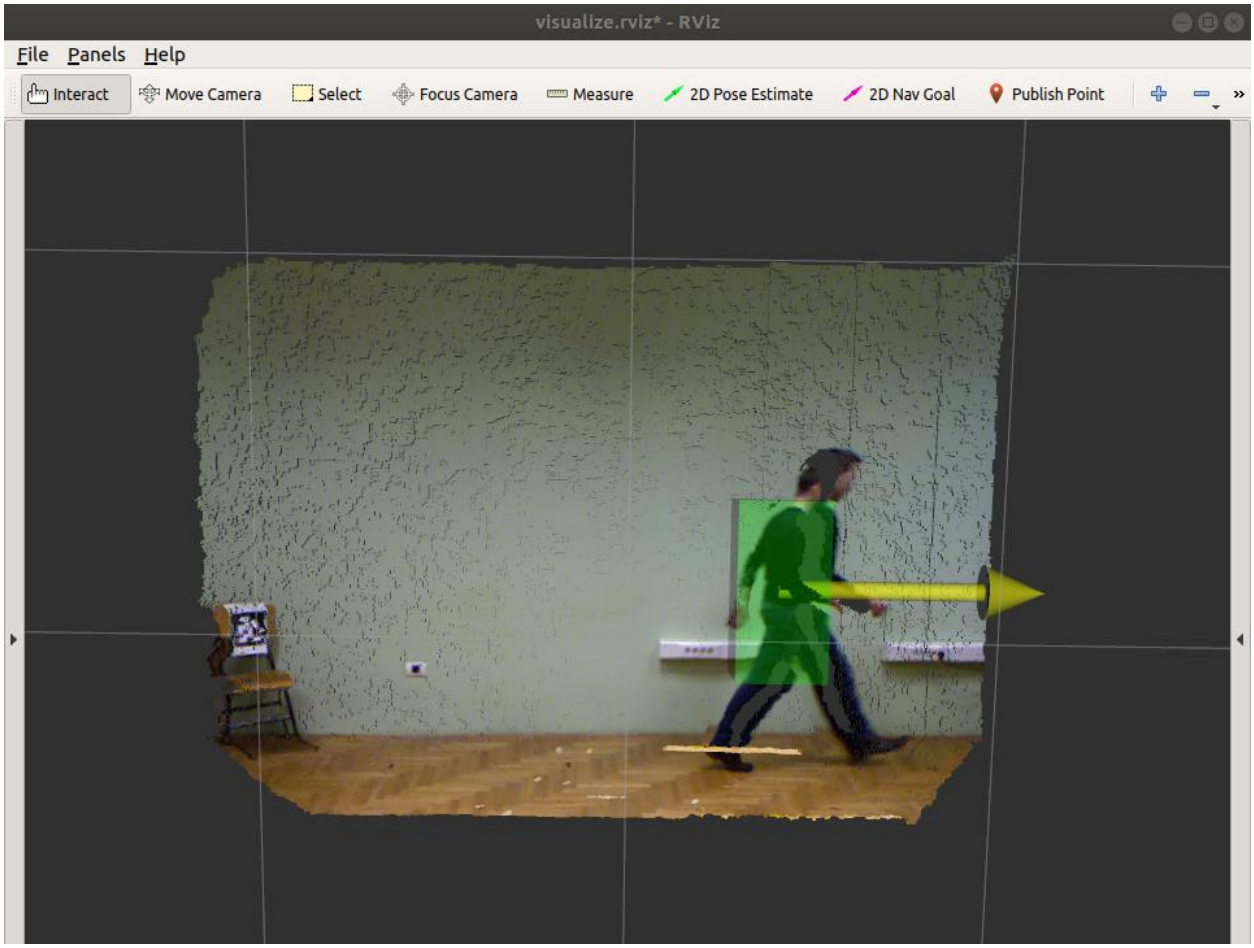
Slika 5.2: 3D vizualizacija

Programski kôd vizualizacije sastoji se od jednog čvora i jedne funkcije povratne veze. U glavnoj funkciji se provodi inicijalizacija čvora, postavljanje i povezivanje pretplatnika. Nakon kreiranja čvora, provjerava se je li omogućena vizualizacija. Ako nije, čvor se gasi. Stvaraju se dva pretplatnika (engl. *Subscribers*) koji se pretplaćuju na podatke 3D oblaka točaka, odnosno podatke o prepoznatim ljudima na sceni. Dva pretplatnika je potrebno sinkronizirati na način da se funkcija povratne veze izvodi ako su podaci oblaka točaka i čovjeka vremenski približno jednaki. Također, stvorena su i dva izdavača (engl. *Publishers*) koja objavljuju informacije oblaka točaka i stvorenih markera za vizualizaciju.

Linija	***Kôd***
1:	`int main(int argc, char** argv)`
2:	`{`
3:	`  // Initialization of the node`
4:	`  ros::init(argc, argv, "VisualizeBbox");`
5:	`  ros::NodeHandle n;`
6:	
7:	`  // Check if visualization is on`
8:	`  bool visualize_flag = false;`
9:	`  n.param("/visualization/ptcloud", visualize_flag, false);`
10:	
11:	`  if (!visualize_flag)`
12:	`    n.shutdown();`
13:	
14:	`  // Subscribers`
15:	`  message_filters::Subscriber<PointCloud2> cloud_sub(n,`
16:	`                                              "PtCloud",`
17:	`                                              2);`
18:	`  message_filters::Subscriber<HumanEntries> entry_sub(n,`
19:	`                                              "HumanData",`
20:	`                                              2);`
21:	
22:	`  // Define synchronizer`
23:	`  approximate_sync_.reset(new ApproximateSync(ApproximatePolicy(4),`
24:	`                                                 cloud_sub,`
25:	`                                                 entry_sub));`
26:	`  // register the 2-element callback`
27:	`  approximate_sync_->registerCallback(boost::bind(&entryCallback,`
28:	`                                                  _1,`
29:	`                                                  _2));`
30:	
31:	`  // publish Pointcloud and markers`
32:	`  markers_pub = n.advertise<visualization_msgs::MarkerArray>("markersHumans", 2);`
33:	`  ptcloud_pub = n.advertise<PointCloud2>("VisualizePtCloud", 2);`
34:	
35:	`  // Enabling the execution of the callback function`
36:	`  ros::spin();`
37:	
38:	`  return 0;`
39:	`}`

Slika 5.3: Čvor za 3D vizualizaciju

Dolaskom informacija o oblaku točaka i prepoznatim ljudima, okida se funkcija povratne veze *entryCallback*, čiji se kôd nalazi u prilogu A (Slika A.1). U funkciji se stvara vektor markera koji će predstavljati ljude. Ulazi se u petlju koja prolazi kroz instance prepoznatih ljudi te za svakog stvara marker oblika kvadra kojeg postavlja na mjesto prepoznatog čovjeka. Također, kako bi bilo moguće vidjeti čovjeka i marker, markeru se postavlja i postotak transparentnosti. Ako se čovjek kreće nekom brzinom, stvara se i marker oblika strelice koji omogućava vizualizaciju smjera brzine čovjeka. Kako bi se mogli prikazati u Rviz alatu, dobiveni vektor markera i oblak točaka se objavljuju na pripadajuće teme. Vizualizaciju s markerima moguće je vidjeti na slici 5.4.



Slika 5.4: 3D vizualizacija s vektorom brzine kretanja

5.2. Upravljanje robotom iz MoveIt-a

5.2.1. Konfiguracija robotskog manipulatora

U ovom radu, upravljanje robotskim manipulatorom ABB-IRB 2400L izvodi se pomoću okvira za planiranje kretanja MoveIt. MoveIt omogućava planiranje trajektorije robota,

vizualizaciju te spajanje sa stvarnim robotskim manipulatorom. Potrebne konfiguracijske datoteke za vizualizaciju, kinematiku i spajanje sa stvarnim manipulatorom preuzete su s GitHub repozitorija Marka Meisela[8]. Preuzeti projekt sastoji se od sljedećih paketa [41]:

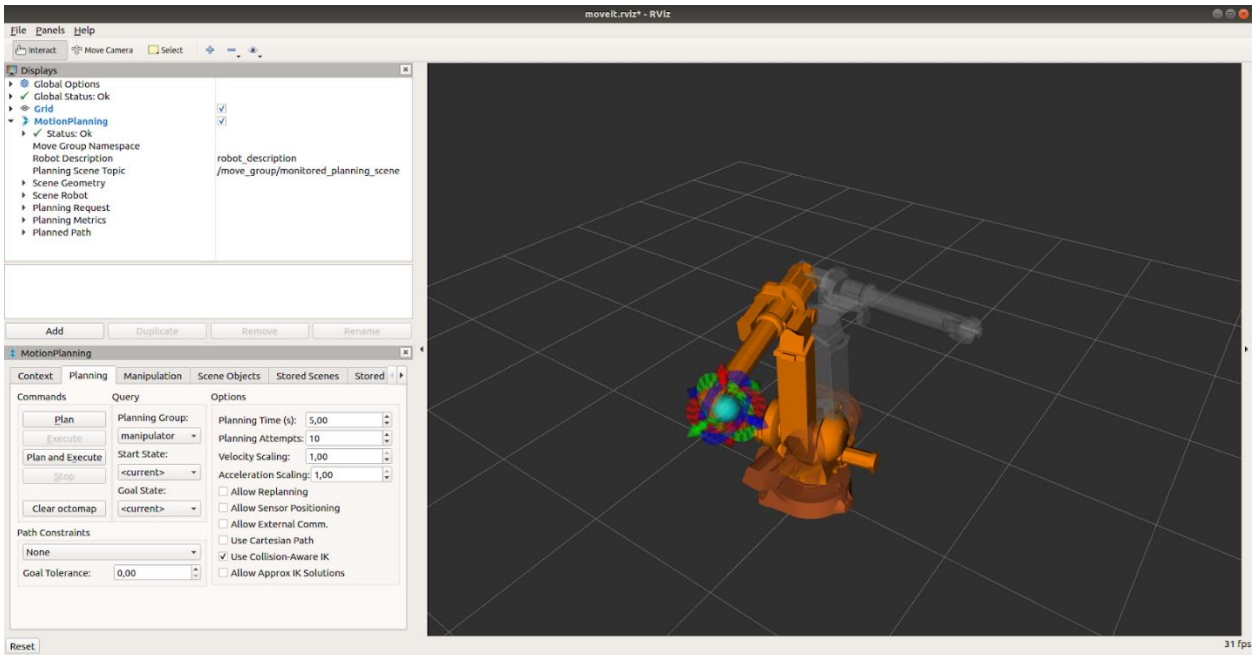
- *abb* – sadrži informacije o autoru i promjenama kroz inačice,
- *abb_driver* – sadrži konfiguracijske datoteke za upravljače robota koje omogućuju komunikaciju s računalom,
- *abb_irb2400_moveit_config* – sadrži datoteke za pokretanje te datoteke za opis mogućnosti manipulatora,
- *abb_irb2400_moveit_plugins* – sadrži datoteke koje opisuju rad direktne i inverzne kinematike manipulatora,
- *abb_irb2400_moveit_support* – sadrži URDF datoteku za opisivanje članaka i zglobova robota te vizualne i kolizijske modele članaka te
- *abb_resources* – sadrži datoteke za opis boje i materijala robotskog manipulatora.

Vizualizacija manipulatora pokreće se naredbom koju je moguće vidjeti na slici 5.5. Pri tome se pali Rviz simulacija u kojoj je moguće pokretati robota pritiskom i povlačenjem kugle na završnom alatu robota (Slika 5.6). Trenutna poza robota prikazana je sivom bojom, a željena poza narančastom bojom.

Linija	***Kod***
1:	`$ roslaunch abb_irb2400_moveit_config demo.launch`

Slika 5.5: Pokretanje simulacije robotskog manipulatora

[8] https://github.com/interpid2/ABB_IRB_2400L



Slika 5.6: Simulacija i vizualizacija manipulatora

Spajanje s pravim robotom moguće je pokretanjem *.launch* datoteke u paketu *abb_irb2400_moveit_config*. Pri pozivanju naredbe, potrebno je predati argumente za onemogućivanje simulacije i IP adresu robota (Slika 5.7).

Linija	***Kod***
1:	`$ roslaunch abb_irb2400_moveit_config moveit_planning_execution.launch sim:=false robot_ip:=172.16.106.185`

Slika 5.7: Povezivanje računala sa stvarnim manipulatorom

Također, potrebno je i povezati se na „mrežu robota“. Fizičko povezivanje s manipulatorom omogućeno je Ethernet kabelom, a u trenutku izrade diplomskog rada, IP adresa robota je 172.16.106.185. Računalo, pomoću kojeg se upravlja manipulatorom, treba biti u istoj podmreži (engl. *subnet*). Primjer konfiguracije za spajanje s manipulatorom se može vidjeti na slici 5.8.

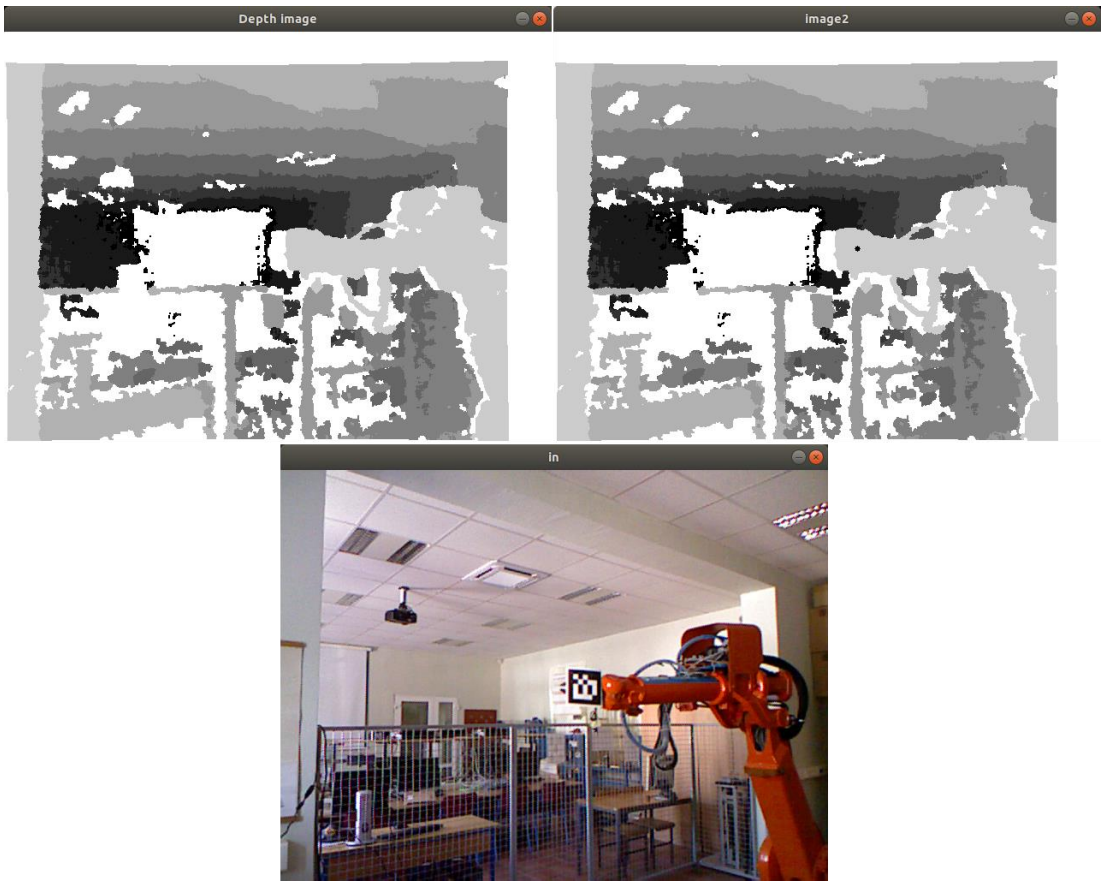
IPv4:
IP address: 172.16.106.185
Netmask: 255.255.255.0
Gateway: 172.16.106.1
DNS and Routes su uključene.

Slika 5.8: Primjer konfiguracije mreže za povezivanje s robotom

5.2.2. Kalibracija Kinect kamere i robota

Prije nego što je moguće povezati sustav za prepoznavanje i praćenje ljudi s manipulatorom, potrebno je kalibrirati Kinect kameru i robota. Kalibracija je odrađena pomoću već gotovog paketa

[42] koji koristi ArUco marker na mjestu završnog alata robota. Tijekom kalibracije, završni alat manipulatora kreće se kroz unaprijed zadani ograničeni prostor ispred kamere s markerom okrenutim prema kameri. Za svaki pomak završnog alata, kamera uslika trenutnu scenu te na slici traži marker. Ukoliko je marker pronađen, završni alat se pomiče i ponovno se slika scena za sljedeću točku. Postupak se ponavlja dok se ArUco marker ne prepozna 10 puta. Postupak kalibriranja moguće je vidjeti na slici 5.9. Rezultat kalibracije predstavlja matrica transformacije ${}^{C}\mathbf{T}_{R}$, odnosno matrica ${}^{R}\mathbf{T}_{C}$, koja opisuje položaj kamere u odnosu na bazu robota i obrnuto.



Slika 5.9: Kalibracija robota i kamere

5.2.3. Uvjeti upravljanja robotskim manipulatorom

Upravljanje kretanjem robotskog manipulatora temelji se na informaciji sustava za prepoznavanje i praćenje ljudi koji daje informaciju o čovjeku prisutnom na sceni. Pomoću te informacije, manipulator usporava, nastavlja kretati se istom brzinom ili se potpuno zaustavlja. Korištene informacije čovjeka predstavljaju udaljenost čovjeka od manipulatora te radijalna i tangencijalna komponenta brzine čovjeka. Postavljeni su uvjeti u kojima se određuje kojom se brzinom završni alat robota treba kretati u ovisnosti o ranije nabrojanim informacijama čovjeka.

Radno područje manipulatora modelirano je pomoću cilindra sa središtem u bazi manipulatora, a polumjer cilindra odgovara dosegu manipulatora. Izraz (5-1) prikazuje promjenu polumjera radnog područja na temelju brzine čovjeka:

$$r = \begin{cases} r_0, & \|\boldsymbol{V}_h\| < 0.1\ \left[\frac{m}{s}\right] \\ 1.1r_0, & 0.1\left[\frac{m}{s}\right] \leq \|\boldsymbol{V}_h\| < 1\ \left[\frac{m}{s}\right] \\ 1.5r_0, & \|\boldsymbol{V}_h\| \geq 1\ \left[\frac{m}{s}\right] \end{cases}, \tag{5-1}$$

gdje r predstavlja polumjer cilindra radnog prostora manipulatora, r_0 predstavlja stvarni doseg robota, a $\|\boldsymbol{V}_h\|$ predstavlja skalarnu vrijednost brzine čovjeka. Brzina kretanja završnog alata manipulatora zadana je sljedećim izrazom:

$$V_{ee} = \begin{cases} 50\%, & inače \\ 50\%/V_t, & 0.1[m/s] \leq \|\boldsymbol{V}_h\| < 2[m/s] \wedge \\ & \left(\|\boldsymbol{V}_t\|(k) - \|\boldsymbol{V}_t\|(k-1)\right) > 0.3 \\ 0\%, & \begin{cases} \|\boldsymbol{V}_h\| \geq 2[m/s] \\ D \leq 15\ [cm] \wedge \|\boldsymbol{V}_r\| > 0 \\ 15 < D\ \leq 30\ [cm] \wedge \angle(\boldsymbol{V}_h, \boldsymbol{V}_r) < 45° \\ D \leq 0 \end{cases} \end{cases}, \tag{5-2}$$

gdje $\angle(\boldsymbol{V}_o, \boldsymbol{V}_r)$ predstavlja kut između vektora brzine čovjeka i vektora radijalne komponente brzine čovjeka, a $\|\boldsymbol{V}_t\|(k) - \|\boldsymbol{V}_t\|(k-1)$ predstavlja apsolutnu razliku između trenutne tangencijalne komponente brzine čovjeka i zadnje pohranjene tangencijalne komponente brzine, a D predstavlja udaljenost čovjeka od radnog prostora robota. Uvjeti predstavljeni izrazima (5-1) i (5-2) implementirani su u programski kôd te se koriste kao osnova upravljanja brzinom kretanja završnog alata manipulatora.

5.2.4. Upravljački algoritam

Programsko rješenje upravljanja manipulatorom nalazi se u radnom prostoru *robot_control_ws* koji sadrži paket *irb2400_control*. Paket *irb2400_control* sadrži čvor *robot_control_node* u kojemu se nalazi cijelo programsko rješenje upravljanja manipulatorom. Izgled programskog kôda čvora se može vidjeti na slici 5.10. Kako bi bilo moguće iskoristiti više funkcija povratnog poziva (engl. *Callback functions*) istovremeno, koristi se klasa *AsyncSpinner*, čijem se konstruktoru tada predaje broj računalnih niti koje je moguće istovremeno iskorištavati za obradu spomenutih funkcija i ostalog kôda. Čvor također omogućava i stvaranje objekta klase *RobotControl* u kojemu se odvija glavnina upravljanja manipulatorom. Programsko rješenje

glavnog dijela upravljanja nalazi se na isječcima kôda u nastavku, a pomoćne funkcije nazale se u prilogu B.

```
Linija  Kôd
1:      #include "irb2400_control/robotcontrol.hpp"
2:      #include <string>
3:
4:      int main(int argc, char** argv)
5:      {
6:          std::string node_name = "robot_control_node";
7:          ros::init(argc, argv, node_name);
8:
9:          ros::NodeHandle nh("");
10:         ros::NodeHandle nh_private("~");
11:
12:         ros::AsyncSpinner spinner(4);
13:         spinner.start();
14:
15:         RobotControlNamespace::RobotControl robot_control_node(nh,
             nh_private);
16:
17:         robot_control_node.loopEverything();
18:
19:         ros::waitForShutdown();
20:     }
```

Slika 5.10: Izgled čvora robot_control_node

Konstruktor klase *RobotControl* kao parametre prima javnu i privatnu referencu na instancu čvora te u inicijalizacijskoj listi postavlja parametre poput zadane pune brzine, stvarnog dosega robota i polumjera cilindra radnog područja i grupu planiranja. Također, poziva se i funkcija *init* u kojoj se postavljaju ostale varijable. Destruktor klase omogućava gašenje dviju instanci čvorova.

```
Linija  Kôd
1:      namespace RobotControlNamespace
2:      {
3:          RobotControl::RobotControl(const ros::NodeHandle &node_handle,
              const ros::NodeHandle &private_node_handle):
4:            nh_(node_handle),
5:            pnh_(private_node_handle),
6:            SPEED_FAST(0.7),
7:            BASE_REACH(1.81),
8:            mediumReach(BASE_REACH*1.1),
9:            highReach(BASE_REACH*1.5),
10:           PLANNING_GROUP("manipulator"),
11:           move_group(moveit::planning_interface::MoveGroupInterface(
                 PLANNING_GROUP))
12:     {
13:         this->init();
14:     }
15:
16:     RobotControl::~RobotControl()
17:     {
18:         nh_.shutdown();
19:         pnh_.shutdown();
20:     }
```

```
...
```

Slika 5.11: Konstruktor i destruktor klase RobotControl

Funkcija *init* ne prima nikakve parametre, a služi samo definiranju određenih varijabli i učitavanju parametara (Slika 5.12). Ovdje se definira pretplatnik na temu */human_tracker_data* koja sadrži informacije dobivene iz sustava za prepoznavanje i praćenje. Pretplatniku se također daje i funkcija povratne veze te veličina međuspremnika koji sprema podatke koji još nisu obrađeni. Veličina spremnika je postavljena na 1 kako bi se osiguralo da sustav uvijek radi sa zadnjim raspoloživim podacima. Nadalje, definira se varijabla koja omogućava spremanje trenutnih vrijednosti zglobova, grupi planiranja se omogućava ponovno planiranje trajektorije te se stvara brojač vremena (engl. *timer*) koji služi kako bi se, nakon izlaska čovjeka iz radnog područja robota, robot nastavio gibati jednakom brzinom kojom se ranije gibao. Također, definiraju se i dva položaja završnog alata između kojih se robot kreće. Kako bi se mogla računati udaljenost robota i čovjeka u odnosu na kameru, učitavaju se ${}^{C}\mathbf{T}_{R}$ i ${}^{R}\mathbf{T}_{C}$ dobivene kalibracijom kamere i robota.

```
Linija  Kôd
1:      void RobotControl::init()
2:      {
3:          // Defining subscribers and publishers
4:          entry_distance_sub = pnh_.subscribe("/human_tracker_data", 1,
                &RobotControl::humanEntriesCallback, this);
5:
6:          joint_model_group = move_group.getCurrentState()->
                getJointModelGroup(PLANNING_GROUP);
7:          move_group.allowReplanning(true);
8:
9:          timer = nh_.createTimer(ros::Duration(3.0),
                                    &RobotControl::timerCallback, this);
10:
11:         //Robot poses
            geometry_msgs::Pose target_pose1;
12:         geometry_msgs::Pose target_pose2;
13:
14:         // pose 1
15:         target_pose1.position.x = 0.735395;
            ...
21:         target_pose1.orientation.z = -3.34027e-05;
22:         // pose 2
23:         target_pose2.position.x = 0.740174;
            ...
29:         target_pose2.orientation.z = 4.3254e-05;
30:
31:         // The robot will iterate through set poses
32:         target_poses.push_back(target_pose1);
33:         target_poses.push_back(target_pose2);
34:         current_target_pose_iterator = target_poses.begin();
35:
36:         // Load transformation matrices
37:         TRC = cv::Mat(4, 4, CV_64F);
```

```
38:       TCR = cv::Mat(4, 4, CV_64F);
39:
40:       RobotControl::loadTransformationMatrices();
41:
42:       robotPosition = cv::Mat(1, 2, CV_64F);
43:       robotPosition.at<double>(0,0) = TRC.at<double>(0, 3);
44:       robotPosition.at<double>(0,1) = TRC.at<double>(1, 3);
45:
46:       breakTrajectoryExecution = false;
47:
48:       CURRENT_SPEED = SPEED_FAST;
49:       calculatedReach = BASE_REACH;
50:
51:       ROS_ERROR("Robot control initialized.");
52:   }
```

Slika 5.12: Funkcija init

Budući da sustav upravljanja robota koristi informacije o čovjeku iz sustava prepoznavanja opisanog u potpoglavlju 5.1, računanja se izvršavaju onom frekvencijom kojom sustav izbacuje rezultate prepoznavanja. Iz tog razloga je računanja potrebno izvoditi u funkcijama povratnih veza. Funkcija povratne veze *humanEntriesCallback* (Slika 5.13) prima informaciju o čovjeku koju daje sustav prepoznavanja te pomoću nje provjerava uvjete dane izrazima (5-1) i (5-2) kako bi prilagodio brzinu robota. Poziva se funkcija *calculateFinalRobotSpeed* koja prima parametar tipa *HumanEntries*, a kao rezultat daje konačnu brzinu robota. Ukoliko ta brzina nije jednaka trenutnoj brzini, trenutno izvođenje trajektorije se prekida, a nova brzina postaje trenutna.

```
Linija  Kôd
1:      void RobotControl::humanEntriesCallback(const
              roi_msgs::HumanEntriesConstPtr& entries_msg)
2:      {
3:          double vRobot =
              RobotControl::calculateFinalRobotSpeed(*entries_msg);
4:
5:          if (vRobot != CURRENT_SPEED)
6:          {
7:              CURRENT_SPEED = vRobot;
8:              breakTrajectoryExecution = true;
9:          }
10:     }
```

Slika 5.13: Funkcija povratne veze humanEntriesCallback

Funkcija *calculateFinalRobotSpeed* (Slika 5.14) na početku postavlja brzinu robota na najveću dozvoljenu brzinu (`SPEED_FAST`) te izračunava doseg radnog područja robota iz dobivenih rezultata prepoznavanja. Nakon toga se prolazi kroz sva prepoznavanja te se popunjavaju matrice komponenata brzine i pozicije čovjeka, na temelju kojih se izračunavaju skalarni iznosi udaljenosti čovjeka od radnog područja, brzine čovjeka i komponenata brzine čovjeka. Nakon toga se prema uvjetima iz izraza (5-2) izračunava brzina manipulatora. Ukoliko je izračunata brzina manja od

trenutne izračunate brzine, nova brzina se uzima kao trenutna. Nakon prolaska kroz sve instance prepoznavanja, funkcija vraća konačnu brzinu manipulatora.

```
Linija  Kôd
1:      double RobotControl::calculateFinalRobotSpeed(roi_msgs::HumanEntries
            entries)
2:      {
3:          vRobot = SPEED_FAST; // max velocity
4:          reach = RobotControl::calculateReach(entries);
5:
6:          cv::Mat vOperatorMat = cv::Mat::zeros(1, 2, CV_64F);
7:          cv::Mat positionOperatorMat = cv::Mat::zeros(1, 2, CV_64F);
8:          cv::Mat vRadialMat = cv::Mat::zeros(1, 2, CV_64F);
9:
10:         for(int i = 0; i < entries.entries.size(); i++)
11:         {
12:             vOperatorMat.at<double>(0,0) = entries.entries[i].Xvelocity;
13:             vOperatorMat.at<double>(0,1) = entries.entries[i].Zvelocity;
14:             positionOperatorMat.at<double>(0, 0) =
                    entries.entries[i].personCentroidX;
15:             positionOperatorMat.at<double>(0, 1) =
                    entries.entries[i].personCentroidZ;
16:
17:             distanceFromRobotEnvelope =
18:                 RobotControl::calculateEuclideanDistance(
                        positionOperatorMat, robotPosition) - reach;

19:             vOperator = calculateOperatorSpeed(vOperatorMat);
20:             vRadial = calculateRadialOperatorSpeed(vOperatorMat,
21:                 positionOperatorMat);
                vTangential = calculateTangentialOperatorSpeed(vOperator,
22:                 vRadial);

23:             vRadialMat = vRadial * (robotPosition - positionOperatorMat)
24:                 / RobotControl::calculateEuclideanDistance(
                        positionOperatorMat, robotPosition);
                radialAngle = calculateRadialAngle(vOperatorMat, vRadialMat);
25:
26:             vTangentialRef = abs(vTangentialRef - vTangential) > 0.3f ?
27:                 vTangential : vTangentialRef;
                vTangentialRef = abs(vTangentialRef) < 0.1f ? 0.1f :
28:                 vTangentialRef;

29:             /* CONDITIONS (2) */
30:             temp_vRobot = SPEED_FAST;
31:
32:             if (vOperator >= 0.1f && vOperator < 2.0f)
33:                 temp_vRobot = SPEED_FAST/(10.0f*vTangentialRef);
34:             else if (vOperator < 0.1f)
35:                 temp_vRobot = 0.5f;
36:             else if (vOperator >= 2.0f || distanceFromRobotEnvelope <
37:                     0.0f)
                {
38:                 temp_vRobot = 0.0f;
39:                 resetTimer = true;
40:             }
41:
```

```
42:             if ((distanceFromRobotEnvelope <= 0.15f && vRadial > 0.0f) ||
43:                 (distanceFromRobotEnvelope > 0.15f &&
44:                  distanceFromRobotEnvelope <= 0.3f && radialAngle <
                     45.0f) ||
                    vOperator >= 2.0f ||
45:                 distanceFromRobotEnvelope < 0.0f)
46:             {
47:                 temp_vRobot = 0.0f;
48:                 resetTimer = true;
49:             }
50:
51:             if (temp_vRobot < vRobot)
52:                 vRobot = temp_vRobot;
53:         }
54:         return vRobot;
55:     }
```

Slika 5.14: Funkcija calculateFinalRobotSpeed

Funkcija *calculateReach* tiče se uvjeta danih u izrazu (5-1), a prikazana je na slici 5.15. Funkcija prima parametar tipa *HumanEntries* te na početku postavlja referentnu veličinu radnog područja na najmanju moguću vrijednost. Zatim se prolazi kroz sve instance prepoznavanja i ispituju se uvjeti koji su opisani spomenutim izrazom. Prolaskom kroz sve instance prepoznavanja, funkcija vraća konačni doseg, odnosno polumjer radnog područja manipulatora.

```
Linija  Kôd
1:      double RobotControl::calculateReach(roi_msgs::HumanEntries entries)
2:      {
3:          double calculatedReach = BASE_REACH;
4:          double referenceReach = BASE_REACH;
5:
6:          cv::Mat vOperatorMat = cv::Mat::zeros(1, 2, CV_64F);
7:
8:          for(int i = 0; i < entries.entries.size(); i++)
9:          {
10:             vOperatorMat.at<double>(0,0) = entries.entries[i].Xvelocity;
11:             vOperatorMat.at<double>(0,1) = entries.entries[i].Zvelocity;
12:
13:             operatorSpeed = calculateOperatorSpeed(vOperatorMat);
14:
15:             if(operatorSpeed < 0.1f)
16:                 calculatedReach = BASE_REACH;
17:
18:             else if(operatorSpeed >= 0.1f && operatorSpeed < 1.0f)
19:                 calculatedReach = mediumReach;
20:
21:             else if(operatorSpeed >= 1.0f)
22:                 return highReach;
23:
24:             if(calculatedReach > referenceReach)
25:                 referenceReach = calculatedReach;
26:         }
27:         return referenceReach;
28:     }
```

Slika 5.15: Funkcija calculateReach

Funkcija *loopEverything* predstavlja najvišu razinu upravljanja manipulatorom jer se u njoj prati kretanje manipulatora i mijenja brzina kretanja. Na početku funkcije se postavlja ograničenje vremena planiranja trajektorije te se definira varijabla tipa *Plan* u koju se sprema izračunata isplanirana trajektorija. Nakon toga se ulazi u *while* petlju koja se vrti sve dok se ne ugasi čvor. U petlji se postavljaju izračunata trenutna brzina kretanja manipulatora i trenutni cilj te se planira trajektorija. Ukoliko dođe do greške i trajektoriju nije moguće isplanirati, funkcijom *assert* se prekida izvođenje programa. U suprotnom se manipulatoru šalje trajektorija te se manipulator počinje kretati. Iduća petlja (*do-while*) omogućava provjeravanje uvjeta za prekidanje izvođenja trajektorije zbog ulaska čovjeka u radno područje. Prekidanjem izvođenja ili završetkom izvođenja trajektorije, izlazi se iz petlje i provjerava se je li izvođenje prekinuto. Ukoliko nije, postavlja se iduća ciljna točka te se program vraća na početak *while* petlje i ponavlja radnje.

Linija	***Kôd***
1:	`void RobotControl::loopEverything()`
	`{`
2:	`    // Setting velocity max planning time`
3:	`    move_group.setPlanningTime(1.0);`
4:	`    moveit::planning_interface::MoveGroupInterface::Plan my_plan;`
5:	`    ROS_ERROR("Initialized plan");`
6:	
7:	`    while(ros::ok())`
8:	`    {`
9:	`        move_group.setMaxVelocityScalingFactor(CURRENT_SPEED);`
10:	
11:	`        geometry_msgs::Pose current_target = *current_target_pose_iterator;`
12:	
13:	`        move_group.setPoseTarget(current_target);`
14:	
15:	`        planning_success = (move_group.plan(my_plan) == moveit::planning_interface::MoveItErrorCode::SUCCESS);`
16:	`        ROS_INFO_NAMED("Plan succession", "Visualizing plan (pose goal) %s", planning_success ? "" : "FAILED");`
17:	`        assert(planning_success);`
18:	
19:	`        move_group.asyncExecute(my_plan);`
20:	
21:	`        do`
22:	`        {`
23:	`            if (resetTimer)`
24:	`            {`
25:	`                resetTimer = false;`
26:	
27:	`                if(timer.hasStarted())`
28:	`                    timer.stop();`
29:	
30:	`                timer.start();`
31:	`            }`
32:	`            else if(timerStopTrigger)`
33:	`            {`
34:	`                timer.stop();`

```
35:                     timerStopTrigger = false;
36:                 }
37:
38:                 if(breakTrajectoryExecution)
39:                 {
40:                     move_group.stop();
41:                     sleep(0.5);
42:
43:                     break;
44:                 }
45:
46:                 current_pose = move_group.getCurrentPose().pose;
47:
48:                 pose_diff_x = abs(
                       (*current_target_pose_iterator).position.x -
                        current_pose.position.x);
49:                 pose_diff_y = abs(
                       (*current_target_pose_iterator).position.y -
                        current_pose.position.y);
50:                 pose_diff_z = abs(
                       (*current_target_pose_iterator).position.z -
                        current_pose.position.z);
51:             } while (pose_diff_x > 0.0005 || pose_diff_y > 0.0005 ||
                    pose_diff_z > 0.0005);
52:
53:             // if the robot finished the trajectory without breaking it
54:             if(!breakTrajectoryExecution)
55:             {
56:                 ++current_target_pose_iterator;
57:
58:                 if(current_target_pose_iterator == target_poses.end())
59:                     current_target_pose_iterator = target_poses.begin();
60:             }
61:             breakTrajectoryExecution = false;
62:         } // while loop end
63:     }
```

Slika 5.16: Funkcija loopEverything

5.2.5. Pokretanje sustava za upravljanje

Upravljanje sustavom pokreće se iz jedne *.launch* datoteke. U ovoj se datoteci pozivaju druge *.launch* datoteke koje pokreću zasebne dijelove sustava. Pozivanje ostalih *.launch* datoteka omogućeno je pomoću paketa *timed_roslaunch* koji omogućava odgađanje pokretanja za određeni broj sekundi. Bitno je omogućiti sustavu dovoljno vremena da se pokrene kako ne bi došlo do kolizije između različitih paketa. Slika 5.17 prikazuje raspored pokretanja dijelova sustava. Najprije se pokreće povezivanje s manipulatorom, a zatim i kamera. Nakon podizanja kamere, podiže se sustav za prepoznavanje ljudi, a na samom kraju i upravljački algoritam manipulatora.

Linija	***Kôd***
1:	`<launch>`
2:	`    <arg name="robot_ip" default="172.16.106.185" />`
3:	
4:	`    <!-- Run Openni for Kinect -->`
5:	`    <include file="$(find timed_roslaunch)/timed_roslaunch.launch">`

```
6:          <arg name="time" value="5" />
7:          <arg name="pkg" value="openni_launch" />
8:          <arg name="file" value="openni.launch" />
9:          <arg name="node_name" value="timed_roslaunch_openni" />
10:     </include>
11:
12:     <!-- Run MoveIt! - robot simulation interface -->
13:     <include file="$(find timed_roslaunch)/timed_roslaunch.launch">
14:         <arg name="time" value="1" />
15:         <arg name="pkg" value="abb_irb2400_moveit_config" />
16:         <arg name="file" value="moveit_planning_execution.launch" />
17:         <!-- <arg name="value" value="sim:=true" /> -->
18:         <arg name="value" value="sim:=false robot_ip:=$(arg
                robot_ip)" />
19:         <arg name="node_name" value="timed_moveit_config" />
20:     </include>
21:
22:     <!-- Run human detection method -->
23:     <include file="$(find timed_roslaunch)/timed_roslaunch.launch">
24:         <arg name="time" value="15" />
25:         <arg name="pkg" value="System_Launch" />
26:         <arg name="file" value="run_kinect_node.launch" />
27:         <arg name="node_name" value="timed_human_tracker" />
28:     </include>
29:
30:     <!-- Run robot trajectory -->
31:     <include file="$(find timed_roslaunch)/timed_roslaunch.launch">
32:         <arg name="time" value="25" />
33:         <arg name="pkg" value="irb2400_control" />
34:         <arg name="file" value="run_RobotControl_node.launch" />
35:         <arg name="node_name" value="timed_robot_control" />
36:     </include>
37: </launch>
```

Slika 5.17: Datoteka za pokretanje sustava

6. EKSPERIMENTALNA EVALUACIJA I REZULTATI

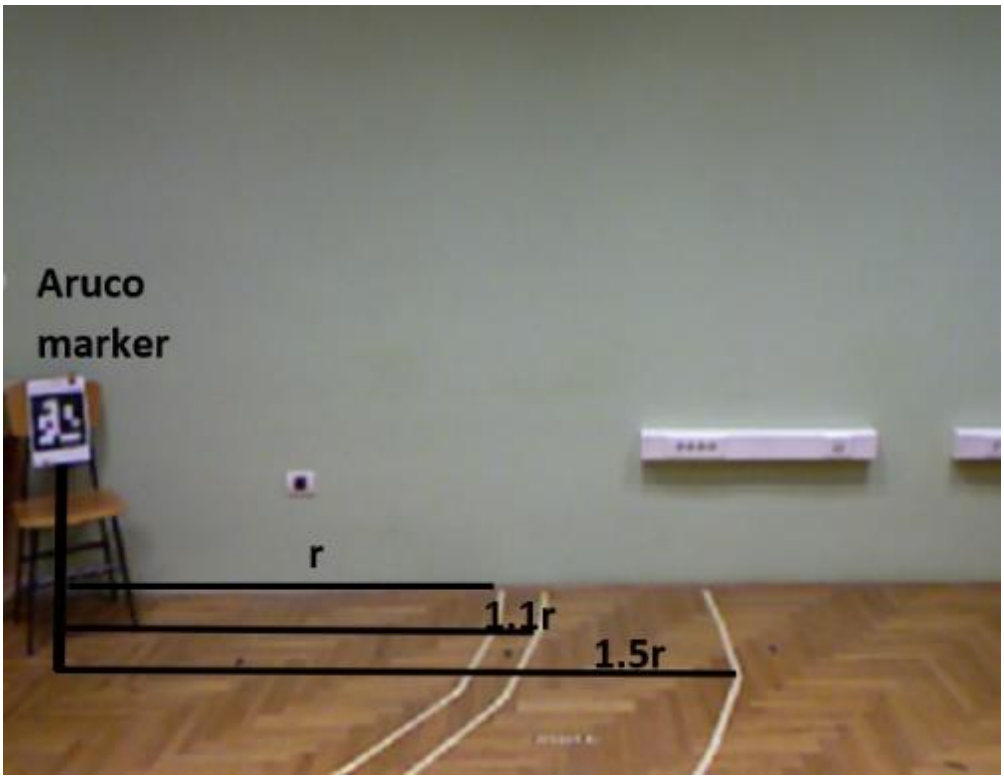
6.1. HDR skup podataka

Kako bi se razvijeni sustav za upravljanje manipulatorom mogao evaluirati, bilo je potrebno snimiti skup podataka (engl. *dataset*) koji će obuhvatiti sve slučajeve u kojima se sustav može naći. Iz tog je razloga snimljen HDR skup podataka, a sustav je na njemu vrednovan. HDR skup podataka čini više videozapisa snimljenih Kinect kamerom. Videozapisi predstavljaju određene situacije u industrijskim pogonima te simuliraju scene u kojima se nalaze robotski sustavi [37]. Nekoliko dijelova skupa podataka, čiji cilj je provjeriti uspješnost sustava za upravljanje robotom, su: test detektora, test predviđanja brzine i test uvjeta za zaustavljanje manipulatora [37]. Skup podataka sadrži i referentne podatke koji čine informacije o broju ljudi na sceni, brzini kretanja ljudi te udaljenosti između robota i čovjeka.

Prostorija u kojoj je snimljen skup podataka simulira industrijsku scenu u kojoj postoji mogućnost interakcije robota i čovjeka. Robotski manipulator predstavljen je ArUco markerom, a dosezi radnog područja manipulatora, predstavljeni izrazom (5-1), označeni su bijelim trakama na podu. Crne trake postavljene na podu međusobno su odvojene 1 m te predstavljaju referencu za hodanje tijekom testova brzine. Opisanu prostoriju za snimanje skupa podataka moguće je vidjeti na slici 6.1. Snimljeni videozapisi spremljeni su kao *rosbag* datoteke, a podaci koje sadržavaju su sljedeći: RGB slika, registrirana dubinska slika, slika dispariteta, 3D oblak točka te informacije o RGB kameri, dubinskoj kameri i projektoru [37].

Testovi detektora uključuju četiri videozapisa: *Više ljudi*, *Kutija 1*, *Kutija 2* i *Kutija 3*. Videozapis *Više ljudi* predstavlja situaciju u kojoj postoji više ljudi na sceni. U ovom slučaju, tri čovjeka ulaze u scenu, nasumično hodaju te se međusobno zaklanjaju, a zatim izlaze iz scene. Videozapisi koji u imenu imaju *Kutija* predstavljaju videozapise u kojima osoba ulazu u scenu noseći kutiju koja zaklanja dio tijela čovjeka. Tri videozapisa uključuju slučajeve kada čovjek ulazi u scenu sa zaklonjenom glavom, kada ulazi u scenu sa zaklonjenom glavom i makne objekt te slučaj kada ulazi u scenu s vidljivom glavom i zaklanja ju na sceni. Testovi predviđanja brzine sadrže videozapise u kojima se čovjek giba brzinom u određenom intervalu. Intervali brzina uključuju: gibanje brzinom većom od 2 m/s, gibanje brzinom između 1 m/s i 2 m/s, gibanje brzinom između 0.1 m/s i 1.0 m/s te gibanje brzinom manjom od 0.1 m/s. Različiti smjerovi kretanja čovjeka u odnosu na kameru u testovima predviđanja brzine omogućavaju provjeravanje uspješnosti određivanja udaljenosti i praćenja, a snimljeni smjerovi kretanja uključuju: kretanje prema kameri,

u smjeru x-osi kamere te dijagonalno u odnosu na kameru. Testovi zaustavljanja robota služe za provjeru uspješnosti cijelog sustava upravljanja robotom. U ovim videozapisima, čovjek na sceni mijenja svoj smjer kretanja i pritom ulazi i izlazi u radno područje robota.



Slika 6.1: Prostorija za snimanje HDR skupa podataka [37]

6.2. Rezultati pokusa

Rezultati pokusa opisanih HDR skupom podataka predstavljeni su tablicama 6.1, 6.2 i 6.3. Korišteni pokazatelji kakvoće sustava su: odziv, preciznost i točnost. Odziv daje mjeru prepoznavanja relevantnih podataka. Maksimizacija vrijednosti odziva je nužna jer sustav mora prepoznati čovjeka u svakom trenutku kako bi se vjerojatnost nesreće minimizirala. Preciznost omogućava uvid u učinkovitost sustava prepoznavanja. Ukoliko sustav izbacuje rezultate ljudi na mjestima gdje ih nema, učinkovitost sustava se smanjuje, stoga je maksimizacija i ove vrijednosti nužna. Točnost omogućava uvid u situacije u kojima je čovjek točno prepoznat (točno pozitivni, engl. *True positives*) te u situacije u kojima čovjek nije lažno prepoznat (točno negativni, engl. *True negatives*) u odnosu na cijeli skup podataka [37]. Točni pozitivi predstavljaju situacije u kojima je čovjek na sceni, a sustav ga je prepoznao i dodijelio mu vrijednost brzine iz pripadajućeg intervala. Točni negativi su situacije u kojima se čovjek ne nalazi na sceni, a sustav ga nije

prepoznao. Netočni pozitivi (engl. *False positives*) opisuju situacije u kojima se čovjek ne nalazi na sceni, a sustav ga je prepoznao i dodijelio mu brzinu, dok su netočni negativi (engl. *False negatives*) situacije u kojima se čovjek nalazi na sceni, ali ga sustav nije prepoznao.

Tablica 6.1 prikazuje rezultate testova prepoznavanja ljudi. Moguće je vidjeti kako je preciznost najveća na videozapisu *Više ljudi*, dok su odziv i točnost maksimalni na videozapisu *Kutija 1*. Problem kod snimki s više ljudi predstavlja zaklanjanje ljudi kada prolaze jedno ispred drugoga. Zbog tog razloga, odziv sustava iznosi 82.84%, što nije dovoljno za industrijske primjene. S druge strane, sustav nema problema u određenim situacijama u kojima je dio čovjeka zaklonjen.

Tablica 6.1: Rezultati testa prepoznavanja ljudi

Naziv videozapisa	Odziv	Preciznost	Točnost
Više ljudi	0.8284	**0.9713**	0.8394
Kutija 1	**0.9792**	0.9592	**0.94**
Kutija 2	0.8601	0.8255	0.7278
Kutija 3	0.88496	0.9434	0.8403

Tablica 6.2 prikazuje rezultate testova predviđanja brzina. Predviđanje brzina uvelike ovisi o sustavu za prepoznavanje ljudi jer na temelju izlaza sustava za prepoznavanje, sustav za praćenje dodjeljuje pripadne identifikacijske oznake ljudima i omogućava izračunavanje brzine svakog čovjeka. Budući da bez posebne opreme nije moguće odrediti točnu brzinu kretanja čovjeka i uspoređivati ju s izlaznom brzinom sustava, kao točna brzina uzima se ona koja se nalazi u promatranom intervalu brzina. Rezultati ovog testa variraju ovisno o brzini kretanja čovjeka. Pri velikoj brzini kretanja čovjeka, metoda ima slabe rezultate jer sustav za prepoznavanje daje loše rezultate, a posljedično je onda i brzina krivo određena. U intervalu brzine između 1 m/s i 2 m/s, rezultati testa variraju ovisno o smjeru kretanja čovjeka. Najbolji rezultat u testu predviđanja brzina postignut je kada se čovjek giba u smjeru x-osi kamere u intervalu između 1 m/s i 2m/s, uključujući odziv, preciznost i točnost. S manjim brzinama kretanja čovjeka, rezultati postaju slabiji. To se događa iz razloga što, iako je čovjek prepoznat na slici, granični okvir koji okružuje čovjeka između instanci prepoznavanja izgubi točnu poziciju. Posljedično, brzina je tada netočno izračunata. Najslabiji rezultat postignut je kada se čovjek giba brzinom manjom od 0.1 m/s u smjeru kretanja prema kameri.

Tablica 6.2: Rezultati testa predviđanja brzina

Naziv videozapisa	Odziv	Preciznost	Točnost
$v \geq 2\ [m/s]$			
Smjer 1	**0.3636**	**1.0**	**0.3636**
$1.0\ [m/s] \leq v < 2.0\ [m/s]$			
Smjer 1	**0.875**	**1.0**	**0.8788**
Smjer 2	0.5484	0.9444	0.5588
Smjer 3	0.7895	**1.0**	0.8
$0.1\ [m/s] \leq v < 1.0\ [m/s]$			
Smjer 1	0.3556	0.6957	0.4194
Smjer 2	**0.7742**	0.96	**0.7576**
Smjer 3	0.6744	**1.0**	0.6744
$v \leq 0.1\ [m/s]$			
Smjer 1	**0.3744**	**1.0**	**0.3744**
Smjer 2	0.2959	**1.0**	0.2959
Smjer 3	0.2610	**1.0**	0.2610

Tablica 6.3 prikazuje rezultate testa zaustavljanja robota, odnosno provjerava se brzina robota u danom trenutku. Moguće je vidjeti kako rezultati testa variraju, a najbolji rezultat postignut je u videozapisu *Test 2*, uključujući odziv, preciznost i točnost.

Tablica 6.3: Rezultati testa zaustavljanja robota

Naziv videozapisa	Odziv	Preciznost	Točnost
Test 1	0.4938	**1.0**	0.6239
Test 2	**0.6429**	**1.0**	**0.7692**
Test 3	0.4839	0.9375	0.6667

7. ZAKLJUČAK

U ovom diplomskom radu predstavljen je sustav upravljanja robotskim manipulatorom zasnovan na prepoznavanju i praćenju ljudi. Rad je nastao u sklopu projekta HDR (engl. *Humans Detected by Robots*), koji je pokrenut u suradnji s tvrtkom Danieli Systec d.o.o. Korišteni robotski sustav uključuje robotski manipulator ABB-IRB 2400L, Microsoft Kinect v1 RGB-D kameru te ROS za upravljanje. Sustav prepoznavanja i praćenja temelji se na AdaBoost algoritmu i stroju s potpornim vektorima, pri tome koristeći HOG deskriptore i Haar značajke. Upravljanje robotskim manipulatorom temelji se na postavljenim uvjetima, a izvodi se pomoću MoveIt okvira za planiranje kretanja. Cjelokupni sustav uspješno je implementiran te ispitan u ROS distribucijama Kinetic i Melodic. Ovakvim sustavima glavna svrha je održavanje sigurnosti. Sigurnost u industrijskim okruženjima vrlo je važna kako bi se izbjegle nesreće, oštećenje strojeva, okruženja, ali i ozljeđivanje ljudi. Kako bi robotski sustav mogao biti primjenjiv u industrijskom okruženju, moraju se postići određene vrijednosti pokazatelja kakvoće. Najstroži zahtjev na sigurnost sustava predstavlja odziv, pokazatelj kakvoće čija vrijednost mora dostići 100% kako bi se osigurala potpuna sigurnost ljudi u okruženju robota. U ovom radu, najveća postignuta vrijednost odziva iznosi 87.5%, što nije dovoljno za primjenu u industriji. Kako bi se osigurala učinkovitost, pokazatelj kakvoće preciznosti treba imati što veću moguću vrijednost. Učinkovitost, u ovom slučaju, označava sposobnost cjelokupnog sustava da omogući robotu što je manje moguće bespotrebnog zaustavljanja i mijenjanja brzine kada to nije potrebno. Najveća postignuta vrijednost preciznosti u ovom radu iznosi 100%, što označava da je moguće postići visoku učinkovitost sustava. Što se tiče točnosti, najveća postignuta vrijednost iznosi 87.88%. Rezultati svih triju pokazatelja kakvoće variraju u ovisnosti o situaciji i vrsti testa. Iako sustav ne ispunjava uvjete odziva za industrijsku primjenu, moguća su poboljšanja sustava kojima bi se rezultat mogao poboljšati. U prvom redu, moguće je snimiti robusniji skup podataka s više primjenjivih situacija. Također, na skupu podataka se mogu poboljšati referentni podaci. Kako bi se mogli poboljšati referentni podaci, potrebna je odgovarajuća oprema uz koju je moguće točno odrediti brzinu čovjeka u svakom pojedinom trenutku. Na robusnom skupu podataka, uz pravilne i precizne anotacije ljudi, moguće je ponovno treniranje modela AdaBoost algoritma i stroja s potpornim vektorima. Korištenjem boljih senzora, može se preciznije odrediti udaljenost čovjeka od kamere, a samim time i točnije odrediti brzina zbog manje varijacija u udaljenosti. Uz bolju opremu, moguća je i fuzija više senzora kako bi se pokrio veći radni prostor i točnije odredio trenutak u kojemu je potrebno zaustaviti robota.

LITERATURA

[1] J. Wu, C. Geyer, i J. M. Rehg, „Real-time human detection using contour cues“, u *2011 IEEE International Conference on Robotics and Automation*, Shanghai, svi. 2011, str. 860–867.

[2] J. Wu, N. Liu, C. Geyer, i J. M. Rehg, „C^4 : A Real-time Object Detection Framework“, lis. 2013

[3] C. Zimmermann, T. Welschehold, C. Dornhege, W. Burgard, i T. Brox, „3D Human Pose Estimation in RGBD Images for Robotic Task Learning“, u *2018 IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, QLD, svi. 2018, str. 1986–1992.

[4] „VoXel | Scientific Volume Imaging“. https://svi.nl/VoXel (pristupljeno lip. 07, 2021).

[5] L. Spinello i K. O. Arras, „People Detection in RGB-D Data“, str. 6.

[6] T. Linder, K. Y. Pfeiffer, N. Vaskevicius, R. Schirmer, i K. O. Arras, „Accurate detection and 3D localization of humans using a novel YOLO-based RGB-D fusion approach and synthetic training data“, u *2020 IEEE International Conference on Robotics and Automation (ICRA)*, Paris, France, svi. 2020, str. 1000–1006.

[7] L. Tian, M. Li, Y. Hao, J. Liu, G. Zhang, i Y. Q. Chen, „Robust 3-D Human Detection in Complex Environments With a Depth Camera“, *IEEE Trans. Multimedia*, sv. 20, izd. 9, str. 2249–2261, ruj. 2018

[8] „Ensemble Methods: Elegant Techniques to Produce Improved Machine Learning Results“, *Toptal Engineering Blog*. https://www.toptal.com/machine-learning/ensemble-methods-machine-learning (pristupljeno svi. 23, 2021).

[9] C. Ninjas, „Boosting in Machine Learning“, *Coding Ninjas Blog*, kol. 14, 2020. https://www.codingninjas.com/blog/2020/08/14/boosting-in-machine-learning/ (pristupljeno svi. 27, 2021).

[10] Y. Freund i R. E. Schapire, „A Short Introduction to Boosting“, str. 14.

[11] R. E. Schapire i Y. Freund, *Boosting: foundations and algorithms*. Cambridge, MA: MIT Press, 2012.

[12] J. D'Souza, „A Quick Guide to Boosting in ML“, *Medium*, ožu. 24, 2018. https://medium.com/greyatom/a-quick-guide-to-boosting-in-ml-acf7c1585cb5 (pristupljeno svi. 27, 2021).

[13] R. E. Schapire, „Explaining AdaBoost“, u *Empirical Inference*, B. Schölkopf, Z. Luo, i V. Vovk, Ur. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, str. 37–52.

[14] „A Guide To Understanding AdaBoost“, *Paperspace Blog*, velj. 23, 2020. https://blog.paperspace.com/adaboost-optimizer/ (pristupljeno svi. 27, 2021).

[15] C. McCormick, „AdaBoost Tutorial · Chris McCormick“. http://mccormickml.com/2013/12/13/adaboost-tutorial/ (pristupljeno svi. 27, 2021).

[16] P. Trunfio, *Service-Oriented Distributed Knowledge Discovery*, sv. 20121229. Chapman and Hall/CRC, 2012.

[17] I. Hrga, „Sveučilište Jurja Dobrile u Puli Fakultet ekonomije i turizma «Dr. Mijo Mirković»“, str. 161.

[18] A. J. Smola i B. Schölkopf, „A tutorial on support vector regression“, *Statistics and Computing*, sv. 14, izd. 3, str. 199–222, kol. 2004

[19] S. Raschka, *Python machine learning: unlock deeper insights into machine learning with this vital guide to cutting-edge predictive analytics*. Birmingham Mumbai: Packt Publishing open source, 2016.

[20] E. K. Nyarko, „STROJ S POTPORNIM VEKTORIMA (engl. SUPPORT VECTOR MACHINE - SVM)“, predavanje iz kolegija *Meko računarstvo*, Osijek, 2020.

[21] C. J. C. Burges, „A Tutorial on Support Vector Machines for Pattern Recognition“, *Data Mining and Knowledge Discovery*, sv. 2, izd. 2, str. 121–167, lip. 1998

[22] „OpenCV: Face Detection using Haar Cascades“. https://docs.opencv.org/master/d2/d99/tutorial_js_face_detection.html (pristupljeno svi. 31, 2021).

[23] BenMauss, „Haar-like Features: Seeing in Black and White“, *Medium*, velj. 17, 2021. https://levelup.gitconnected.com/haar-like-features-seeing-in-black-and-white-1a240caaf1e3 (pristupljeno svi. 31, 2021).

[24] „Histogram of Oriented Gradients explained using OpenCV“, pros. 06, 2016. https://learnopencv.com/histogram-of-oriented-gradients/ (pristupljeno svi. 31, 2021).

[25] „Feature Descriptor | Hog Descriptor Tutorial“, *Analytics Vidhya*, ruj. 04, 2019. https://www.analyticsvidhya.com/blog/2019/09/feature-engineering-images-introduction-hog-feature-descriptor/ (pristupljeno svi. 31, 2021).

[26] R. K. McConnell, „Method of and apparatus for pattern recognition“, US4567610A, sij. 28, 1986 Pristupljeno: svi. 31, 2021. [Na internetu]. Dostupno na: https://patents.google.com/patent/US4567610A/en

[27] N. Dalal i B. Triggs, „Histograms of Oriented Gradients for Human Detection“, u *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, San Diego, CA, USA, 2005, sv. 1, str. 886–893.

[28] B. Jahaj, „Primjena histograma orijentiranih gradijenata za detekciju objekata“, info:eu-repo/semantics/bachelorThesis, Josip Juraj Strossmayer University of Osijek. Faculty of

Electrical Engineering, Computer Science and Information Technology Osijek. Department of Communications. Chair of Multimedia Systems and Digital Television, 2020. Pristupljeno: svi. 31, 2021. [Na internetu]. Dostupno na: https://urn.nsk.hr/urn:nbn:hr:200:863828

[29] H. Bristow i S. Lucey, „Why do linear SVMs trained on HOG features perform so well?“, *arXiv:1406.2419 [cs]*, lip. 2014, Pristupljeno: svi. 31, 2021. [Na internetu]. Dostupno na: http://arxiv.org/abs/1406.2419

[30] M. Munaro, C. Lewis, D. Chambers, P. Hvass, i E. Menegatti, „RGB-D Human Detection and Tracking for Industrial Environments“, u *Intelligent Autonomous Systems 13*, Cham, 2016, str. 1655–1668.

[31] „ROS.org | About ROS“. https://www.ros.org/about-ros/ (pristupljeno lip. 10, 2021).

[32] J. Lentin i J. Cacace, *Mastering ROS for Robotics Programming: Design, build, and simulate complex robots using the Robot Operating System*, Second Edition. Birmingham: Packt Publishing Limited, 2018.

[33] M. Quigley, B. Gerkey, i W. D. Smart, *Programming Robots with ROS: A Practical Introduction to the Robot Operating System*. 2016.

[34] „Metapackages - ROS Wiki“. http://wiki.ros.org/Metapackages (pristupljeno lip. 11, 2021).

[35] „ABB IRB 2400L - Robots Done Right“. https://robotsdoneright.com/ABB/2000-Series/ABB-IRB-2400L.html (pristupljeno lip. 08, 2021).

[36] „ABB IRB 2400L“, *RobotWorx*. https://www.robots.com/robots/abb-irb-2400l (pristupljeno lip. 08, 2021).

[37] D. Mihelčić, D. Svirac, V. Šimundić, P. Đurović, i R. Cupec, „Safety System for Industrial Robots Based on Human Detection Using an RGB-D Camera“, u *Međunarodni skup za informacijsku, komunikacijsku i elektroničku tehnologiju*, Opatija, ruj. 2021, str. 8. (u procesu objavljivanja)

[38] A. Hartley, „Microsoft: Xbox Kinect release date 10 November“, *TechRadar*. https://www.techradar.com/news/gaming/microsoft-xbox-kinect-release-date-10-november-710465 (pristupljeno lip. 10, 2021).

[39] Z. Zhang, „Microsoft Kinect Sensor and Its Effect“, *IEEE Multimedia*, sv. 19, izd. 2, str. 4–10, velj. 2012

[40] M. Špičko, „Konstrukcija dvoosnog mehanizma robotske glave“, final_project_of_the_undergraduate_study, Sveučilište u Zagrebu, Fakultet strojarstva i brodogradnje, 2015. Pristupljeno: lip. 10, 2021. [Na internetu]. Dostupno na: http://repozitorij.fsb.hr/3205/

[41] M. Meisel, „Sustav upravljanja robotskim manipulatorom pomoću 3D kamere u ROS okviru“, info:eu-repo/semantics/masterThesis, Josip Juraj Strossmayer University of Osijek. Faculty of Electrical Engineering, Computer Science and Information Technology Osijek. Department of Computer Engineering and Automation. Chair of Automation and Robotics, 2018. Pristupljeno: lip. 15, 2021. [Na internetu]. Dostupno na: https://urn.nsk.hr/urn:nbn:hr:200:974415

[42] M. Tošeski, „Kalibracija robotske ruke i kamere za robotsku manipulaciju zasnovanu na vizualnom servoingu“, Josip Juraj Strossmayer University of Osijek. Faculty of Electrical Engineering, Computer Science and Information Technology Osijek. Department of Computer Engineering and Automation. Chair of Automation and Robotics, 2017. Pristupljeno: lip. 23, 2021. [Na internetu]. Dostupno na: https://urn.nsk.hr/urn:nbn:hr:200:606415

SAŽETAK

U ovom diplomskom radu, predstavljen je sustav za upravljanje robotskim manipulatorom temeljen na prepoznavanju i praćenju ljudi. Rad je nastao kao dio projekta HDR (engl. *Humans Detected by Robots*) u suradnji s tvrtkom Danieli Systec d.o.o. Korišteni sustav za prepoznavanje i praćenje ljudi temelji se na AdaBoost algoritmu i stroju s potpornim vektorima, koristeći HOG deskriptore i Haar značajke. Robotski sustav sastoji se od robotskog manipulatora ABB-IRB 2400L, RGB-D kamere Microsoft Kinect v1 te ROS razvojnog okvira. Cjelokupni sustav implementiran je i ispitan na ROS Kinetic i ROS Melodic distribucijama. Eksperimentalna evaluacija provedena je na HDR skupu podataka, pri čemu su računati pokazatelji kakvoće odziv, preciznost i točnost. Provedena evaluacija odnosi se na testiranje detektora, predviđanja brzina i zaustavljanja robota. Najveća postignuta vrijednost odziva iznosi 87.5%, dok najveće vrijednosti preciznosti i točnosti iznose 100%. Rezultati variraju u ovisnosti o situaciji i vrsti testa.

Ključne riječi: AdaBoost, HDR, prepoznavanje ljudi, robotski manipulator, ROS, SVM

ABSTRACT

Title: Security system for human detection in the robot workspace based on AdaBoost algorithm and support vector machines

In this thesis, a system for controlling robotic manipulators based on human detection and tracking is presented. The thesis was written as a part of the HDR (*Humans Detected by Robots*) project in cooperation with Danieli Systec. The system used for human detection and tracking is based on the AdaBoost algorithm and Support Vector Machines (SVM), using HOG descriptors and Haar features. The robotic system consists of an ABB-IRB 2400L robotic manipulator, a Microsoft Kinect v1 RGB-D camera, and ROS development framework. The entire system was implemented and tested on ROS Kinetic and ROS Melodic distributions. Experimental evaluation was performed on the HDR dataset with recall, precision, and accuracy used as metrics. The evaluation includes detector tests, speed prediction tests, and robot stopping tests. The highest achieved value of recall was 87.5%, while the highest value of precision and accuracy was 100%. The results vary depending on the situation and the type of test.

Keywords: AdaBoost, HDR, human detection, robotic manipulator, ROS, SVM

ŽIVOTOPIS

Valentin Šimundić rođen je 17.3.1998. godine u Osijeku. U Đakovu pohađa osnovnu školu Josipa Antuna Ćolnića te gimnaziju Antuna Gustava Matoša. 2016. godine upisuje preddiplomski studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, nakon čijeg završetka na istom fakultetu upisuje diplomski studij računarstva, smjer Robotika i umjetna inteligencija. Dobitnik je STEM stipendije na preddiplomskom studiju, državne stipendije za posebne skupine studenata na diplomskom studiju, Rektorove nagrade za rad na projektu HDR te nagrade za uspješnost u studiranju na diplomskom studiju. Također, ima rad prihvaćen za objavljivanje na međunarodnoj znanstvenoj konferenciji MIPRO 2021 pod nazivom *Safety System for Industrial Robots Based on Human Detection Using an RGB-D Camera*.

PRILOG

PRILOG A – DODATNE FUNKCIJE SUSTAVA ZA PREPOZNAVANJE I PRAĆENJE LJUDI

Linija	*Kôd*
1:	`void entryCallback(const PointCloud2ConstPtr& cloud_msg, const HumanEntriesConstPtr& entry_msg)`
2:	`{`
3:	`  // Create array for markers`
4:	`  visualization_msgs::MarkerArray pplMarkers;`
5:	`  pplMarkers.markers.clear();`
6:	
7:	`  // Loop through message data, create bbox markers and push on back`
8:	`  // of array`
9:	`  for (int i = 0; i < entry_msg->entries.size(); i++)`
10:	`  {`
11:	`    visualization_msgs::Marker personMarker;`
12:	`    personMarker.id = 2 * i;`
13:	`    personMarker.header.frame_id = "/camera_rgb_optical_frame";`
14:	`    personMarker.header.stamp = ros::Time::now();`
15:	
16:	`    personMarker.ns = "person";`
17:	
18:	`    personMarker.type = visualization_msgs::Marker::CUBE;`
19:	`    personMarker.action = visualization_msgs::Marker::ADD;`
20:	
21:	`    personMarker.pose.position.x =`
22:	`        entry_msg->entries[i].personBoundingBoxTopCenterX;`
23:	`    personMarker.pose.position.y =`
24:	`        entry_msg->entries[i].personBoundingBoxTopCenterY;`
25:	`    personMarker.pose.position.z =`
26:	`        entry_msg->entries[i].personBoundingBoxTopCenterZ;`
27:	
28:	`    personMarker.pose.orientation.x = 0.0;`
29:	`    personMarker.pose.orientation.y = 0.0;`
30:	`    personMarker.pose.orientation.z = 0.0;`
31:	`    personMarker.pose.orientation.w = 0.0;`
32:	
33:	`    personMarker.scale.x = entry_msg->entries[i].ROIwidth;`
34:	`    personMarker.scale.y = entry_msg->entries[i].ROIheight;`
35:	`    personMarker.scale.z = entry_msg->entries[i].ROIwidth;`
36:	
37:	`    // color.a represents alpha - transparency`
38:	`    personMarker.color.a = 0.28;`
39:	`    personMarker.color.r = 0.0;`
40:	`    personMarker.color.g = 1.0;`
41:	`    personMarker.color.b = 0.0;`
42:	
43:	`    personMarker.lifetime = ros::Duration(0.25);`
44:	
45:	`    // velocity length`
46:	`    double vX = entry_msg->entries[i].Xvelocity;`
47:	`    double vY = entry_msg->entries[i].Zvelocity;`
48:	`    double velocityVal = sqrt(vX * vX + vY * vY);`
49:	
50:	`    // if velocity is greater than 0, create a velocity marker`
51:	`    if (velocityVal > 0)`

```
52:       {
53:
54:         geometry_msgs::Point start_point;
55:         geometry_msgs::Point end_point;
56:
57:         start_point.x =
58:           entry_msg->entries[i].personBoundingBoxTopCenterX;
59:         start_point.y =
60:           entry_msg->entries[i].personBoundingBoxTopCenterY;
61:         start_point.z =
62:           entry_msg->entries[i].personBoundingBoxTopCenterZ;
63:
64:         end_point.x =entry_msg->entries[i].personBoundingBoxTopCenterX
65:                      + vX;
66:         end_point.y =entry_msg->entries[i].personBoundingBoxTopCenterY;
67:         end_point.z =entry_msg->entries[i].personBoundingBoxTopCenterZ
68:                      + vY;
69:
70:         // velocity Marker
71:         visualization_msgs::Marker velocityMarker;
72:         velocityMarker.type = visualization_msgs::Marker::ARROW;
73:         velocityMarker.action = visualization_msgs::Marker::ADD;
74:         velocityMarker.id = 2 * i + 1;
75:         velocityMarker.header.frame_id = "/camera_rgb_optical_frame";
76:         velocityMarker.header.stamp = ros::Time::now();
77:
78:         velocityMarker.ns = "velocity";
79:
80:         velocityMarker.points.push_back(start_point);
81:         velocityMarker.points.push_back(end_point);
82:
83:         velocityMarker.scale.x = 0.1;
84:         velocityMarker.scale.y = 0.3;
85:
86:         // color.a represents alpha - transparency
87:         velocityMarker.color.a = 0.6;
88:         velocityMarker.color.r = 1.0;
89:         velocityMarker.color.g = 1.0;
90:         velocityMarker.color.b = 0.0;
91:
92:         velocityMarker.lifetime = ros::Duration(0.25);
93:         pplMarkers.markers.push_back(velocityMarker);
94:       }
95:       pplMarkers.markers.push_back(personMarker);
96:     }
97:
98:     // Publish data
99:     markers_pub.publish(pplMarkers);
100:    ptcloud_pub.publish(cloud_msg);
101:  }
```

Slika A.1: Funkcija povratne veze entryCallback

PRILOG B – DODATNE FUNKCIJE UPRAVLJAČKOG ALGORITMA

```
Linija  Kôd
1:      void RobotControl::loadTransformationMatrices()
2:      {
3:          // Find the calibration package
4:          std::string matrices_path = ros::package::getPath("abb2400_sim");
5:          std::string TRC_filename_path = matrices_path + "/TRC.txt";
6:          std::string TCR_filename_path = matrices_path + "/TCR.txt";
7:
8:          std::ifstream matrix_file;
9:          matrix_file.open(TRC_filename_path);
10:
11:         // Read every element from matrix
12:         for (int i = 0; i < 4; i++)
13:         {
14:             for (int j = 0; j < 4; j++)
15:             {
16:                 matrix_file >> TRC.at<double>(i, j);
17:             }
18:         }
19:         matrix_file.close();
20:
21:         matrix_file.open(TCR_filename_path);
22:         for (int i = 0; i < 4; i++)
23:         {
24:             for (int j = 0; j < 4; j++)
25:             {
26:                 matrix_file >> TCR.at<double>(i, j);
27:             }
28:         }
29:         matrix_file.close();
30:     }
```

Slika B.1: Funkcija loadTransformationMatrices

```
Linija  Kôd
1:      double RobotControl::calculateRadialOperatorSpeed(cv::Mat vOperator,
                cv::Mat positionOperator)
2:      {
3:          // Assuming that the velocity of the base of the robot is 0
4:          /**
5:           * all are vectors except vR (which is a scalar)
6:           *
7:           *       (vOperator - vRobot) * (pOperator - pRobot)
8:           * vR = ---------------------------------------------
9:           *                  |pOperator - pRobot|
10:          */
11:         // norm (with type NORM_L2) calculates the Euclidean distance
12:            between two points
13:         return vOperator.dot(robotPosition - positionOperator) /
               RobotControl::calculateEuclideanDistance(robotPosition,
                 positionOperator);
14:     }
```

Slika B.2: Funkcija calculateRadialOperatorSpeed

Linija	*Kôd*
1:	`double RobotControl::calculateRadialAngle(cv::Mat vOperator,`
	`cv::Mat vRadial)`
2:	`{`
3:	`double vOp_mag = (pow(vOperator.at<double>(0,0), 2) +`
	`pow(vOperator.at<double>(0,1), 2));`
4:	`double vRad_mag = (pow(vRadial.at<double>(0,0), 2) +`
	`pow(vRadial.at<double>(0,1), 2));`
5:	`double magnitudeOfVectors = sqrt(vOp_mag * vRad_mag);`
6:	`return acos(vOperator.dot(vRadial) / magnitudeOfVectors) *`
	`(180.0f/CV_PI);`
7:	`}`

Slika B.3: Funkcija calculateRadialAngle

Linija	*Kôd*
1:	`double RobotControl::calculateTangentialOperatorSpeed(double`
	`vOperator, double vRadial)`
2:	`{`
3:	`return vOperator*vOperator - vRadial*vRadial;`
4:	`}`

Slika B.4: Funkcija calculateTangentialOperatorSpeed

Linija	*Kôd*
1:	`double RobotControl::calculateOperatorSpeed(cv::Mat vOperator)`
2:	`{`
3:	`return sqrt(vOperator.at<double>(0,0)*vOperator.at<double>(0,0) +`
	`vOperator.at<double>(0,1)*vOperator.at<double>(0,1));`
4:	`}`

Slika B.5: Funkcija calculateOperatorSpeed

Linija	*Kôd*
1:	`double RobotControl::calculateEuclideanDistance(cv::Mat position1,`
	`cv::Mat position2)`
2:	`{`
3:	`double eucl_x = pow(position1.at<double>(0,0) -`
	`position2.at<double>(0,0), 2);`
4:	`double eucl_y = pow(position1.at<double>(0,1) -`
	`position2.at<double>(0,1), 2);`
5:	
6:	`return sqrt(eucl_x + eucl_y);`
7:	`}`

Slika B.6: Funkcija calculateEuclideanDistance

Linija	*Kôd*
1:	`void RobotControl::timerCallback(const ros::TimerEvent& event)`
2:	`{`
3:	`    breakTrajectoryExecution = true;`
4:	`    timerStopTrigger = true;`
5:	`}`

Slika B.7: Funkcija timerCallback