Vizijski sustav za praćenje poze ljudskog tijela

Javor, Ante

VISOKA TEHNIČKA ŠKOLA U BJELOVARU
STRUČNI STUDIJ MEHATRONIKE

VIZIJSKI SUSTAV ZA PRAĆENJE POZE LJUDSKOG TIJELA

Završni rad br. 14/MEH/2017

Ante Javor

Bjelovar, rujan 2017.

VISOKA TEHNIČKA ŠKOLA U BJELOVARU
STRUČNI STUDIJ MEHATRONIKE

VIZIJSKI SUSTAV ZA PRAĆENJE POZE LJUDSKOG TIJELA

Završni rad br. 14/MEH/2017

Ante Javor

Bjelovar, rujan 2017.

obrazac ZR - 001

Visoka tehnička škola u Bjelovaru

Trg E. Kvaternika 4, Bjelovar

1. DEFINIRANJE TEME ZAVRŠNOG RADA I POVJERENSTVA

Kandidat: **Javor Ante** Datum: 14.06.2017. Matični broj: 001180

JMBAG: 0314011529

Kolegij: **OSNOVE PROGRAMIRANJA**

Naslov rada (tema): **Vizijski sustav za praćenje poze ljudskog tijela**

Područje: **Tehničke znanosti** Polje: **Računarstvo**

Grana: **Programsko inženjerstvo**

Mentor: **Zoran Vrhovski, mag.ing.el.techn.inf.** zvanje: **viši predavač**

Članovi Povjerenstva za završni rad:

1. **dr.sc. Alan Mutka, predsjednik**
2. **Zoran Vrhovski, mag.ing.el.techn.inf., mentor**
3. **dr.sc. Igor Petrović, član**

2. ZADATAK ZAVRŠNOG RADA BROJ: 14/MEH/2017

U radu je potrebno:
1. opisati mogućnosti biblioteke OpenCV u svrhu računalnog vida
2. opisati funkcije biblioteke OpenCV korištene za praćenje poze ljudskog tijela
3. izraditi i opisati vizijski sustav za praćenje poze ljudskog tijela
4. opisati korištenu softversku i hardversku podršku za praćenje poze ljudskog tijela
5. izraditi i opisati programski kod za skeletonizaciju ljudskog tijela

Zadatak uručen: 14.06.2017. Mentor: **Zoran Vrhovski, mag.ing.el.techn.inf.**

Zahvala

Zahvaljujem se roditeljima, obitelji i djevojci na podršci tijekom studija.

SADRŽAJ

1. UVOD

Cijene kamera su u padu, a njihova kvaliteta u stalnom je rastu. Današnje kamere mogu snimati video zapis u 4K[1] rezoluciji i to na mobilnim uređajima koji su danas svakodnevica. U isto vrijeme snaga računala, odnosno mogućnost obrade velike količine podataka (*ng. big data),* sve je dostupnija. Upravo takav napredak i dostupnost kamera omogućava razvoj dijela računarstva koji se bavi računalnim vidom (*eng. computer vision*) te je i trend korištenja računalnog vida u osjetnom porastu. Sukladno tome razvijaju se i algoritmi računalnog vida koji su zbog manje kvalitete kamera, računalnih resursa i kompleksnosti čekali smanjenje tehnoloških barijera. Upravo takve algoritme čine i algoritmi za praćenje čovjeka.

Praćenje čovjeka i njegovih navika uvijek je bila interesantna tema znanstvenicima koji se bave humanističkim i društvenim znanostima, međutim i programeri su pronašli nešto zanimljivo u praćenju čovjeka. Korištenjem računalnog vida mogu se pratiti pozicija tijela, gestikulacije, lice pa čak i osjećaji koje čovjek prenosi putem svoga tijela. Sve to otvara veliki prostor za inovacije u medicini, robotici, sportu, društvenim aktivnostima, komunikaciji s računalima i slično. Algoritmi za praćenje čovjeka zbog svoje kompleksnosti razvijaju se već dugi niz godina što ih čini jednim od najduže razvijanih algoritama u segmentu računalnog vida. Razlog tome je što su svi ljudi fizičkih različiti, različito se odijevaju i ponašaju, dok je tehnički vrlo zahtjevno stvoriti 3D sliku iz 2D izvora. Ako se zanemari razlika u korištenju različite opreme kao što su dubinske kamere ili više kamera, pristup praćenju ljudskog tijela moguće je podijeliti u dva segmenta: s markerima i bez marker. Kroz završni rad napravljena je analiza oba pristupa te su predstavljeni rezultati istih.

Razvoj algoritama za praćenje čovjeka [1, 2, 3, 4] je u uzlaznoj putanji, što predstavlja pozitivnu stvar i put prema rješavanju problema praćenja čovjeka. Iako trenutno još ne postoji univerzalno rješenje koje rješava problem praćenja čovjeka u većini slučajeva, razvijaju se usko specijalizirani sustavi koji djelomično rješavaju problem praćenja pozicije ljudskog tijela. Samim tim i ovaj završni rad se bavi specijaliziranim sustavom u kojem je cilj mjeriti duljinu ljudskih udova uz pomoć markera i bez markera. Specijalizirani sustavi danas se primjenjuju i razvijaju ovisno o potrebama i željama tržišta.

[1] 4K označava rezoluciju pri kojoj je broj horizontalnih piksela oko 4000 a vertikalnih oko 2000.

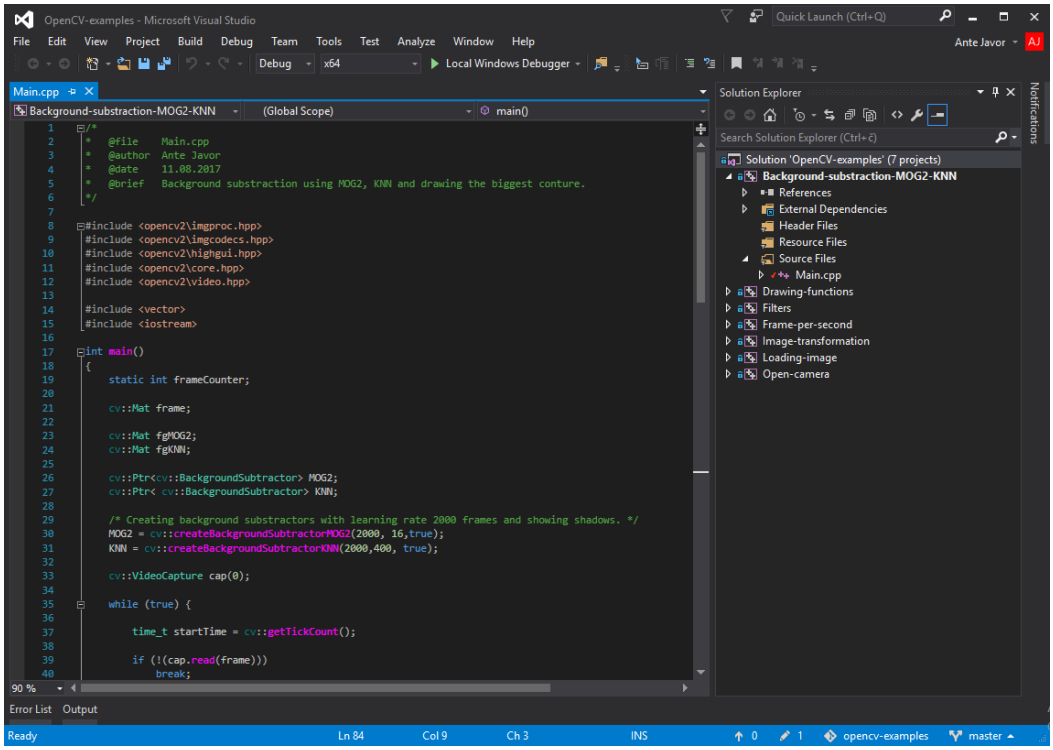
Završni rad počinje s analizom korištenog softvera, nakon čega slijedi uvod u računalni vid. U poglavlju 4 obrađena je analiza popularne računalne biblioteke (eng. *library)* koja se bavi računalnim vidom (eng. *Open Source Computer Vision*) koja je korištena za razvijanje sustava praćenje poze ljudskog tijela. Opisana je i dodatna analiza funkcija i algoritama koji su korišteni za izradu rješenja. Rješenje koje se koristi za praćenje poze čovjeka s markerima i bez markera opisano je u poglavlju 5.

2. KORIŠTENA PROGRAMSKA PODRŠKA U RAZVOJU VIZIJSKOG SUSTAVA ZA PRAĆENJE POZE LJUDSKOG TIJELA

2.1. *Visual Studio Community*

Visual Studio Community 2015 je proizvod tvrtke *Microsoft* i jedno od razvojnih okruženja namijenjeno studentima, manjim grupama programera i zajednici otvorenog koda (eng. *open source*). Inačica okruženja *Community* prikazana na slici 2.1. besplatna je i može se preuzeti s web stranice tvrtke *Microsoft. Visual Studio Enterprise* te *Visual Studio Professional* inačice su razvojnih okruženja koje se naplaćuju i služe za razvoj komercijalnog softvera. Razvojno okruženje koristi se za razvoj (eng. *development*), otklanjanje pogrešaka (eng. *debug*), testiranje i suradnju. Pomoću razvojnog okruženja *Visual Studio Community 2015* moguće je razviti aplikacije za: *Windows* operativne sustave, mobilne operativne sustave, web i aplikacije u oblaku. Osim aplikacija moguće je razvijati i igrice, alate za paket *Microsoft Office*, baze podataka, ekstenzije za razvojna okruženja i slično.

Razvojno okruženje je u svrhu ovog završnog rada korišteno za razvoj i otklanjanje pogrešaka konzolne aplikacije koja obuhvaća algoritme za obradu slike.



Slika 2.1. Prozor razvojnog okruženja *Visual Studio Community*

2.2. *Cmake*

Cmake[2] je aplikacija koja služi za upravljanje procesom stvaranja softvera. Ova aplikacija je besplatna i otvorenog je koda. Za stvaranje softvera zahtjeva samo C++ kompajler (eng. *compiler*). *Cmake* je korišten u svrhu stvaranja izvršnog koda biblioteke *OpenCV*, kako bi se ista mogla koristiti za stvaranje novih projekata. Izvorni kod preuzet s interneta obrađuje se pomoću aplikacije *Cmake* kako bi se isti prilagodio projektu i kako bi se generirale iskoristive biblioteke.

2.3. *GitHub*

GitHub[3] je web aplikacija za kontrolu razvoja softvera koju uglavnom koriste programeri za rad na izvornom kodu i dokumentaciji. Aplikaciji *GitHub* moguće je pristupiti kao privatna osoba ili kao organizacija, a količina dozvoljenih repozitorija, njihova dostupnost i ostalo ovisit će o tome koristi li se besplatni ili plaćeni pristup platformi. *OpenCV* se razvija na platformi *GitHub* gdje cijela zajednica programera i developera može pristupiti izvornom kodu (eng. *source code*) i slati zahtjeve za izmjene i dopune. U isto vrijeme postoje i programeri koje te dopune i izmjene odobravaju, odnosno odbijaju.

Izvorni *OpenCV* kod korišten za sustav praćenja pozicije ljudskog tijela u ovom radu preuzet je s platforme *GitHub*. Ondje je moguće preuzeti verziju s ažurnim promjenama koje se događaju svaki dan, dok se na službenoj stranici biblioteke izdanje ažurira samo kada dođe do većih promjena. Na *GitHub* platformi nalazi se i dio biblioteke koji se ne nalazi u glavnom izdanju. Radi se o neovisnim dodatnim modulima koje je razvila zajednica. Ti moduli ne ulaze u glavno izdanje jer slični algoritmi u njemu već postoje ili nisu dovoljno općeniti. U ovom projektu korišten je jedan takav modul pa je izvorni *OpenCV* kod preuzet s platforme *GitHub* nakon čega su moduli generirani pomoću softvera *Cmake*. Više informacija o dodatnim modulima bit će u poglavlju 4..

[2] Za više informacija otići na službenu stranicu Cmake *https://cmake.org/*.

[3] Za više informacija otići na službenu stranicu GitHub *https://github.com/*.

3. RAČUNALNI VID

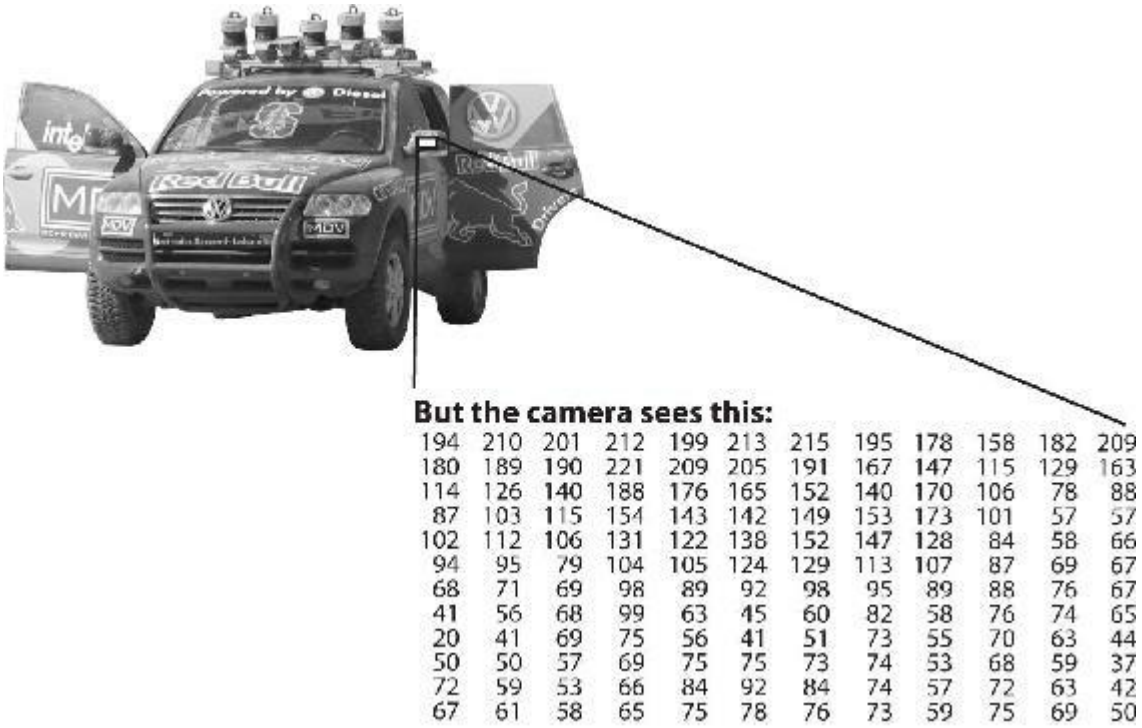
Računalni vid je područje računalstva u kojem se analizom i obradom slike može doći do informacija koje mogu pomoći donositi odluke o ponašanju tehničkog sustava. Na primjer, može dati odgovore na sljedeća pitanja:

- nalazi li se na slici automobil ili lopta,
- koja je pozicija automobila ili lopte,
- koje su boje automobil ili lopta.

Pod obradom slike *(eng. image processing)* podrazumijeva se obrada digitalnih podataka u obliku slika i videa iz kojih je moguće dobiti informaciju o obliku, poziciji, orijentaciji, veličini, dimenziji i boji slično.

Kada čovjek gleda sliku ili video, on može jasno raspoznati objekte, udaljenosti tih objekata, boje i pozicije. Cilj digitalne obrade je doći do svih tih informacija. Međutim, ljudski mozak ima veliku sposobnost obrade podataka i informacija koje svakodnevno prima. Naučen je kako razmišljati i pohraniti informacije. Osim dobre obrade podataka, čovjek ima na raspolaganju oči koje predstavljaju senzor u procesu obrade slike. Oči čovjeka automatski se prilagođavaju svjetlu i funkcioniraju jednako dobro u svim uvjetima osvjetljenja. Kod tehničkih sustava je upravo suprotno. Oni nisu naučeni da kroz godine obrađuju informacije, već im programeri funkcionalnost zadaju kroz algoritme. Između ostalog, kamera kao senzor daleko je lošija od ljudskog oka jer nema automatski fokus i prilagodbu osvjetljenju te često generira šum (*eng. nosie),* a može imati i distorziju. Ti nedostaci predstavljaju prepreku za rješavanje problema računalnog vida. Izvor informacija o vanjskom svijetu je matrica brojeva koja zapravo predstavlja sliku. Računalo sliku vidi kao strukturiranu matricu brojeva koju je potrebno obraditi. Brojevi u matrici koji predstavljaju sliku mogu u sebi imati superponiran šum koji je potrebno otkloniti [5]. Na slici 2.1. nalazi se prikaz načina na koji računalo vidi dio slike, retrovizor automobila. Problem šuma i distorzije je malen kada se usporedi s problemom stvaranja i rekonstruiranja 3D objekata iz 2D slike. Problem rekonstrukcije upravo je jedan od najzanimljivijih i najzahtjevnijih zadataka računalnog vida te za njega ne postoji univerzalno rješenje. Kada se u to sve ubroje faktori šuma, distorzije, osvjetljenja i kretanja može se zaključiti da su problemi računalnog vida vrlo zahtjevni. Unatoč problemima koji se javljaju pri

obradi slike, postoje razna rješenja za uklanjanje šuma u slici, distorzije slike i ostalog. Ova rješenja bit će opisana u narednim poglavljima.



Slika 2.1. Kako računalo vidi digitalni zapis slike [5]

Obzirom na dugi razvoj algoritama za praćenje čovjeka, razvijeno je nekoliko različitih načina praćenja čovjeka. Svi načini mogu ponuditi djelomično rješenje problema, ali ne i stabilno rješenje u svim uvjetima rada. Iz toga razloga razvijaju se specijalizirani sustavi koji u određenim uvjetima osvjetljenja, pozicije kamere, postavljenim markerima i sličnim uvjetima mogu ponuditi traženo i najbolje rješenje.

4. OPEN SOURCE COMPUTER VISION

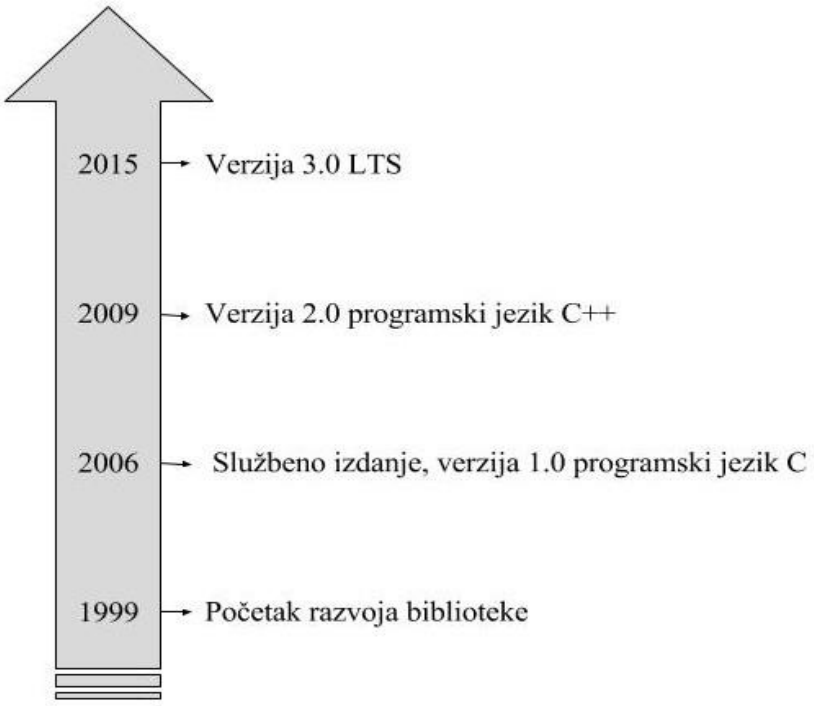
4.1. Opće informacije o biblioteci *OpenCV*

OpenCV (eng. *open source computer vision library*) je računalna biblioteka koja je dostupna na (*http://opencv.org*). Pokrenuta je 1999. godine od strane Intela, čiji je cilj bio razvoj računalnog vida i umjetne inteligencije. Biblioteka pruža temelje računalnog vida i dostupna je svima. Napisana je u programskim jezicima C i C++ te podržava operativne sustave *Linux*, *Windows* i *Mac OS X* operativne sustave. Trenutno su u razvoju sučelja (eng. *interface*) za programske jezike: *Pyton, Java* i *MATLAB*. U razvoju je i podrška za *Android* i *iOS* mobilne operativne sustave. Razvoju biblioteke *OpenCV* najviše su doprinijeli *Intel* i *Google*, a posebno *Itseez* kojega je nedavno otkupio *Intel* [6].

Biblioteka *OpenCV* dizajnirana je s naglaskom na performansama, a sve to kako bi se mogla koristiti i u najzahtjevnijim slučajevima. Optimizirana je u programskom jeziku C++ i podržava višejezgrene procesore odnosno višenitno izvođenje (eng. *multithread*). *OpenCV* biblioteka je razvijana s ciljem da bude laka za korištenje i dostupna svima. To omogućava korisnicima da razvijaju napredne algoritme i rješavaju probleme računalnog vida brzo i učinkovito. *OpenCV* sadrži nekoliko stotina algoritama i funkcija koji su primjenjivi u mnogim poljima kao što su medicina, sigurnost, robotika, industrijska proizvodnja, sport i drugo. Potrebno je izdvojiti da biblioteka sadrži i moderan pristup rješavanju problema računalnog vida i to strojnim učenjem (ng. *machine learning*).

Obzirom da je *OpenCV* otvorenog koda, svi dijelovi biblioteke mogu se koristiti za razvoj komercijalnih aplikacija potpuno besplatno. Licenca je besplatna pa je zajednica koja koristi biblioteku velika, a intenzivno je koriste kompanije kao što su *Microsoft, Intel, SONY, Simens, Google, IBM* i slične [5, 6].

Prema [5, 6] *OpenCV* ima preko 14 milijuna preuzimanja i preko 47 tisuća aktivnih korisnika. Dnevno se koristi u milijunima mobilnih uređaja, industrijskim pogonima, mjerenjima, autonomnim vozilima, dronovima, sustavima za mješovitu i virtualnu stvarnost i slično. *OpenCV* 3.0 i novije verzije razvijane su modularnim pristupom te su kompletno napisane u programskom jeziku C++. Na slici 4.1 moguće je vidjeti vremenski razvoj biblioteke. Trenutni razvoj biblioteke podignut je na novu razinu uz platformu za kontrolu razvoja softvera *GitHub*.



Slika 4.1 Vremenski razvoj biblioteke *OpenCV*

Zbog velike popularnosti biblioteke i njezinog razvoja, dolazi do velike količine uspješnih i korisnih algoritama koje je potrebno održavati pa je iz toga razloga napravljeno neovisno skladište (eng. *repository*) pod nazivom „opencv_contrib“. Taj dio biblioteke napravljen je i održavan od strane korisnika biblioteke *OpenCV* koji pritom prate stil, dokumentaciju i testiranje glavnog dijela biblioteke.

Trend popularnosti računalnog vida je u stalnom rastu pa se i glavna biblioteka *OpenCV* mora razvijati sukladno tržištu. Prema autoru [5] u budućnosti će se raditi na nekima od sljedećih smjernica: dubinskom učenju, mobilnim platforma, naočalama za virtualnu i mješovitu stvarnost, ugradbenim aplikacijama, dubinskim kamerama, robotici, rješenjima u oblaku i *online* edukacijama.

4.2. Moduli biblioteke *OpenCV*

OpenCV podijeljen je na module od verzije 2.0 pa nadalje. Razlog tome je što je inicijalna verzija 1.0 imala sučelje u programskom jeziku C, nakon čega *OpenCV* ima sučelje u programskom jeziku C++ . To im omogućava neovisnost i modularnost pa su stvari za razvoj biblioteke pojednostavljene i modernizirane.

Moduli koji čine verziju 3.2.0 koja je korištena za razvoj sustava za praćenje čovjeka su:

- *core* - modul koji sadrži sve osnove tipove podataka, njihove operatore i osnovne funkcije. Ovaj modul proteže se kroz sve druge module u biblioteci.
- *imgproc* - modul u kojem se nalaze funkcije za transformaciju slike, filtre i pretvorbe slike.
- *imgcodecs* - modul koji sadrži funkcije za čitanje i pohranjivanje slika.
- *videoio* - modul koji sadrži klase i funkcije za čitanje i pohranjivanje videa.
- *highgui* - modul koji sadrži funkcije za korisničko sučelje, prikaz slike i videa.
- *video* - modul koji se koristi za analizu i obradu videa. U njemu su sadržane klase za oduzimanje pozadine *(*eng. *background substraction*) koja se koristi u sustavu za praćenje pozicije ljudskog tijela.
- *calib3d* - modulu u kojem se nalaze algoritmi za kalibraciju kamera i 3D rekonstrukciju.
- *features2d* - modul koji obuhvaća algoritme za detektiranje, obradu, označavanje i pretraživanje ključnih točaka sa slike.
- *objdetect* - modulu u kojem se nalaze algoritmi za detektiranje objekata kao što su ljudi, lica, registarske oznake i slično. Sadrži i algoritme koje je moguće trenirati da prepoznaju neke zadane objekte.
- *ml* - modul koji sadrži mnoštvo algoritama za strojno učenje koji su prilagođeni da rade s tipovima podataka i objekata iz biblioteke *OpenCV*.
- *flann* – modulu u kojem se nalaze metode za pretragu i obradu višedimenzionalnih podataka na veoma učinkovit način. Većina tih metoda koristi se u drugim funkcijama biblioteke *OpenCV* te se rijetko koristi direktno.

- *photo* - modul koji sadrži alate za računalnu fotografiju, kao što su HDR[4] upravljanje, uklanjanje šuma, obrada detalja i slično.
- *stitching* - modul koji se koristi za spajanje, kalibraciju i obradu višestrukih slika u jednu.
- *cuda* - moduli koji počinju s prefiksom „cuda“ zapravo čine već nabrojane osnovne module koji su prilagođeni obradi na grafičkim procesorskim jedinicama. Iako postoji i jedan dio modula koji su pisani isključivo za upotrebu na grafičkim procesorskim jedinicama.

4.3. Tipovi podataka

Kako bi korištenje funkcija i svih algoritama koje obuhvaća *OpenCV* bilo što jednostavnije i brže, definirani su osnovni tipovi podataka poput: *Vec, Mat, Scalar, Point* i *Rect*. U narednim poglavljima biti će navedene ključne klase i funkcije korištene u sustavu za praćenje pozicije ljudskog tijela. Prilikom navođenja funkcija naglasak je dan samo na važnim argumentima funkcije, odnosno onima koji su ključni za izvršenje funkcije.

4.3.1. Klasa *Point*

Klasa *Point* jedna je od jednostavnijih u biblioteci *OpenCV*, a se često koristi za pohranjivanje pozicije na slici, odnosno piksela. U programskom kodu 4.1 prikazano je kako se definira objekt tipa *Point*.

Programski kod 4.1. Stvaranje klase *Point*

```
/* Definicija objekta pomoću konsturktora. */

cv::Point pointInt {1,3}
cv::Point2d pointDouble {3.12, 4.56}
cv::Point3i point3Int {12,5,4}

/* Pristup podatkovnim članovima objekta point3int.*/

point3Int.x() // iznosi 12
point3Int.y() // iznosi 5
point3Int.z() // iznosi 4
```

[4] HDR (*eng. High dynamic range)* je kratica za tehnologiju koja predstavlja poboljšanja u kvaliteti kontrasta slike.

Analizom izvornog koda iz primjera 4.1 moguće je primijetiti da se funkcijama, objektima i konstantama u biblioteci *OpenCV* pristupa unutar imenika (eng. *namespace*) *cv::*. Imenici se koriste kako ne bi došlo do zagađivanja globalnog područja. Kada se radi na velikom projektu s velikom količinom funkcija i objekata, imenici služe kako ne bi došlo do sukoba imena. Stoga i standardna biblioteka predložaka (eng. *Standard Template Library*) programskog jezika C++ koristi vlastiti imenik *std::*. Isto tako iz koda se može zaključiti da će zadnje slovo u nazivu klase označiti kakav će tip podatka spremiti objekt *Point* (*double, int*, ili *float).* Podatkovnim članovima objekta moguće je pristupiti preko javno dostupnih funkcija *x(), y()* i z() koje kao rezultat vraćaju vrijednost istih.

4.3.2. Klasa *Mat*

Klasa *Mat* je glavni tip podatka koji se koristi u biblioteci *OpenCV*. Gotovo sve funkcije primaju, odnosno vraćaju objekt tipa *Mat* ili u svojoj unutarnjoj implementaciji koriste taj tip objekta. Radi se o tipu koji može pohraniti višedimenzionalne nizove. Međutim, svaki element u nizu ne mora biti jedna vrijednost nego može sadržavati više vrijednosti, odnosno imati više kanala. Ne ulazeći u detalje unutarnje implementacije, tip podatka *Mat* je idealan za pohranjivanje onoga što je najpotrebnije za biblioteku *OpenCV*, a to su slike u digitalnom zapisu. Između ostalog, posjeduje i veliku količinu funkcija za manipulaciju samim objektom. U programskom kodu 4.2. prikazan je kod za stvaranje objekta tipa *Mat.*

Programski kod 4.2. Definicija objekta *Mat*

```
/* Deklaracija objekta */
cv::Mat matrica;

/* Stvaranje i definicija objekta */
matrica.create(10, 10, CV_64FC3);

/* Drugi način pomoću konsturktora i tipa cv::Scalar */
cv::Mat matricaNova( 10, 10, CV_64FC3, cv::Scalar(0.0f, 0.0f, 0.0f));
```

Analizom koda iz primjera 4.2. moguće je zaključiti da se u prvom primjeru koristi funkcija klase *Mat* za stvaranje objekta. Funkcija ima nekoliko inačica, a to znači da funkcije koriste isto ime, ali se razlikuju po svome potpisu, odnosno po tipu argumenata u deklaraciji. Takav pristup zove se preopterećenje funkcije *(*eng. *function overloading)* i omogućuje korištenje različitih argumenta prilikom stvaranja objekta *Mat.* U ovom slučaju prvi argument je broj redaka, drugi argument je broj stupaca, dok je treći argument tip podatka koji element pohranjuje. U drugom načinu pomoću konstruktora klase *Mat* stvoren je objekt. Početna tri argumenta identična su

prvom pristupu, dok je u četvrtom argumentu proslijeđen tip *Scalar* kako bi svi elementi triju kanala imali vrijednost 0.0. Tip *Scalar* sličan je tipu *Point* uz razliku da pohranjuje tipove *float* i *double* te ima jednu dimenziju više.

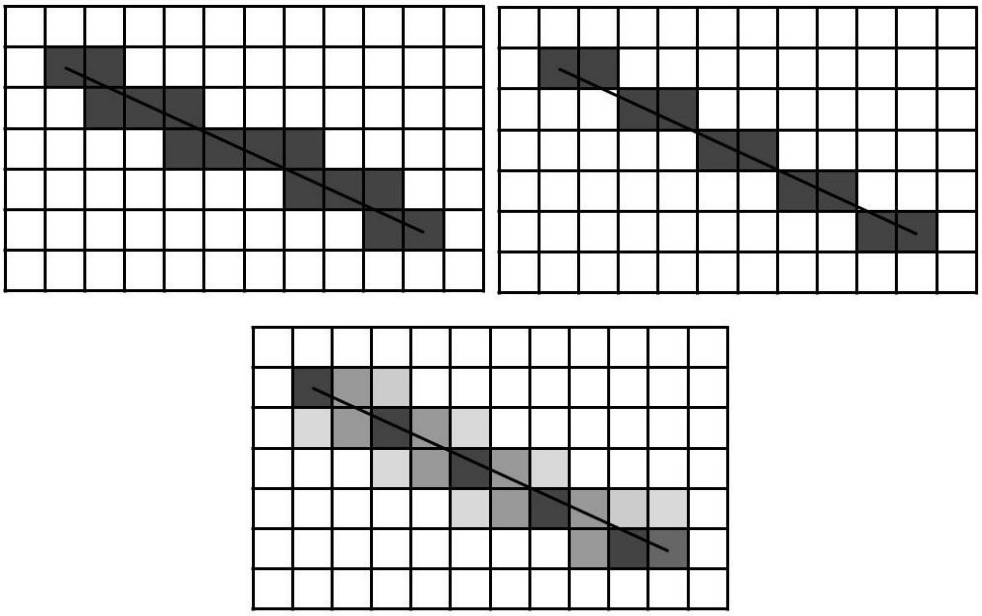
Važno je napomenuti da prilikom stvaranja objekta *Mat* podaci nisu vezani za sami objekt, nego je moguće prenositi podatke s jednog objekta na drugi, a da se pritom podaci ne dupliraju. Unutarnja implementacija klase *Mat* koristi mehanizme pokazivača i referenci kako bi se štedjeli računalni resursi prilikom preslikavanja i obrade slika.

4.4. Funcije za crtanje i iterator

Kako određeni rezultati obrade zahtijevaju vizualizaciju, moguće je koristiti funkcije za crtanje raznih oblika i teksta. *OpenCV* posjeduje nekoliko takvih funkcija za crtanje linije, kružnice, pravokutnika, elipse, a one se nalaze u modulu *imgproc*. U projektu su korištene funkcije za crtanje linije, višestrukih linija, funkcije za crtanje kontura i funkcije za tekst.

4.4.1. Funkcija *line()*

Funkcija *line()* crta liniju između dvije točke. Kao parametar, ova funkcija prima sliku na koju treba crtati liniju, točku gdje počinje i završava linija te debljinu, boju i tip linije. Tip će odrediti algoritam crtanja odnosno spajanja piksela. Na slici 4.2. prikazane su tri moguće opcije spajanja linije: *4-connected, 8-connected* i *anti-aliased.*



Slika 4.2. Rezultati algoritama za crtanje [5]

Kod *4-connected* algoritma susjedni pikseli se spajaju vertikalno i horizontalno, dok se kod *8-connected* algoritma mogu spajati vertikalno, horizontalno i dijagonalno. Treća opcija koristi Gaussov filtar o kojemu će biti govora kasnije. U programskom kodu 4.3. prikazan je primjer korištenja funkcije *line()*.

Programski kod 4.3. Poziv funkcije *line()*

```
/* Priprema argumenata */
cv::Mat slika {600, 600, CV_32FC3};
cv::Scalar boja {255.0, 255.0, 255.0};
cv::Point start {0, 0};
cv::Point kraj {600, 600};
int debljina = 3;
int algoritam = 8;

/* Poziv funkcije i crtanje linije */
cv::line(slika, start, kraj, boja, debljina, algoritam);
```

Na početku programskog koda priprema se slika po kojoj će se crtati linija. Slika je rezolucije 600 x 600 piksela. Svaki piksel ima tri kanala tipa *float.* Svaki kanal sadrži vrijednost boje u spektru boja RBG[5] (crvena, plava i zelena boja). Važno je napomenuti da argument boja tipa *Scalar* čita redom komponente plave, zelene i crvene boje. U programskom kodu 4.3 vrijednost boje je bijela. Nakon definiranja boje, slijedi definiranje točki i debljine linije u pikselima te algoritma spajanja piksela.

4.4.2. Funkcija *polylines()*

Funkcija *polylines()* ima nekolicinu istih argumenata kao funkcija *line(),* a to su: slika, boja, debljina linije i algoritam spajanja piksela. Primarni cilj funkcije *polylines()* je crtanje neke zatvorene konture. Primjerice, ako je cilj uokviriti neki detektirani objekt i slično, sama funkcija ima nekoliko svojih inačica. U ovom završnom radu funkcija se koristi za crtanje višestrukih linija, a kao argumente, osim spomenutih, prima i *vektor* tipa *Point.* Vektor definira redoslijed kojim je potrebno spojiti točke i *bool* argument koji definira je li potrebno spojiti prvu i zadnju točku iz niza.

4.4.3. Funkcija *putText()*

Funkcija *putText()* je glavna i jedina funkcija crtanja teksta. Ona prima iste argumente koje primaju već spomenute funkcije. Uz njih prima i argument tipa *string* koji sadrži željeni tekst,

[5] RGB je kratica koja na engleskom jeziku; *red, green i blue*.

argument tipa *Point* koji označuje početak teksta, konstantu za tip fonta i argument tipa *double* za veličinu teksta.

4.4.4. Klasa *LineIterator*

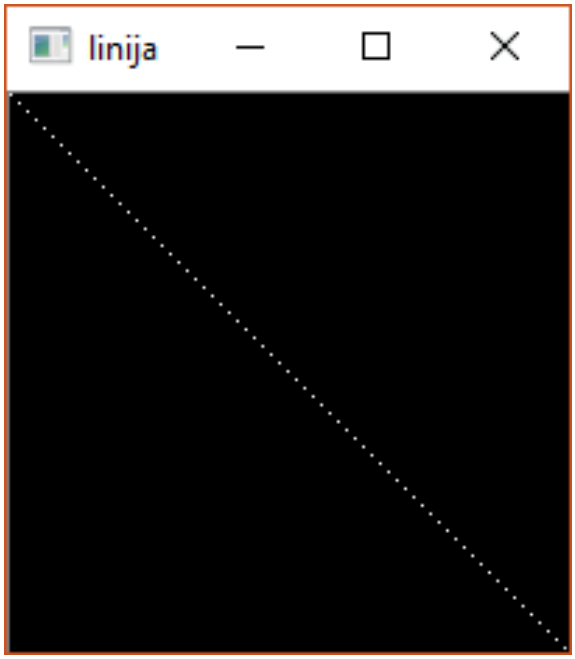
Iterator na razumljiv način omogućava pristup strukturiranim podacima. Pomoću iteratora moguće je prolaziti kroz podatke te ih pretraživati, sortirati i mijenjati. *OpenCV* ima svoj operator za linije. Radi se o iteratoru koji omogućuje micanje po liniji odnosno njezinim pikselima. O iteratoru će biti govora u kasnijim poglavljima. Treba napomenuti da kao argumente prima početak i kraj linije tipa *Point*, algoritam spajanja, sliku i *bool* varijablu koja označava s koje strane počinje. *LineIterator* ima korisne operatore kao što su operator inkrementiranja i operator dereferenciranja. Programski kod 4.4. prikazuje program koji koristi klasu *LineIterator* za mijenjanje svakog trećeg piksela linije u bijelu boju.

Programski kod 4.4. Primjer upotrebe klase *LineIterator* za promjenu boje piksela

```
#include <opencv2\opencv.hpp>
int main()
{
   /* Priprema argumenata */
   cv::Mat slika{ 200, 200, CV_32FC3 };
   cv::Point start{ 0, 0 };
   cv::Point kraj{ 200, 200 };

   /* Stvaranje iteratora i petlja. */
   cv::LineIterator iterator(slika, start, kraj, 8, true);
   for (int i = 0; i < iterator.count; i++)
   {
          if (i % 3 == 0)
                 slika.at<cv::Vec3f>(iterator.pos())=cv::Vec3f(255.0,255.0,255.0);
          iterator++;
   }
   cv::imshow("linija", slika);
   cv::waitKey(0);
   return 0;
}
```

Program počinje naredbom *#include* koja uključuje sve module koje *OpenCV* obuhvaća. U uvjetima izvođenja *for* petlje treba primijetiti da je korištena javno dostupna varijabla objekta *.count*. Varijabla vraća ukupan broj piksela koje taj objekt obuhvaća, a to je duljina zadane linije. Nakon toga za svaki treći piksel linije potrebno je boju promijeniti u bijelu, što se ostvaruje pomoću funkcije *.at()* klase *Mat.* Funkcija omogućava pristup pojedinom pikselu slike kada joj se proslijede koordinate piksela. Koordinate piksela dobivene su pomoću funkcije *.pos()* klase *LineIterator* koja vraća poziciju na kojoj se iterator nalazi. Na kraju programa nalaze se još dvije

funkcije: funkcija za prikaz slike *imshow()* i funkcija koja čeka unos korisnika s tipkovnice *waitKey()*. O njima će biti govora u narednim poglavljima. Rezultat izvođenja programskog koda 4.4 prikazan je na slici 4.3.



Slika 4.3. Rezultat izvođenja programskog koda 4.4.

4.5. Funkcije za rukovanje slikama i videom

Kako bi rukovanje slikama i videom bilo što jednostavnije, kao i razmjena informacija između operativnog sustava i hardvera, *OpenCV* ima funkcije koje su posvećene upravo tome. Uz funkcije za spremanje, obradu i učitavanja slika i videa, biblioteka nudi još neke elemente za razvoj korisničkog sučelja, obradu događaja i datoteka. Ovdje će biti objašnjene samo funkcije korištene u sustavu za praćenje pozicije ljudskog tijela.

4.5.1. Funkcije *imread() i imwrite()*

Kao što imena ovih funkcija otkrivaju, radi se o funkcijama koje se koriste za čitanje i pohranjivanje slika. Obje funkcije primaju argument tipa *string* koji predstavlja ime slike ili putanju do iste, dok kao drugi argument funkcija *imread()* prima konstantu koja predstavlja na koji način sliku treba učitati. Drugi parametar funkcije *imwrite()* je slika koju je potrebno pohraniti.

4.5.2. Klasa *VideoCapture*

Klasa *VideoCapture* nudi rješenja za čitanje, pohranjivanje i kontrolu videa. Ona sadrži nekoliko funkcija i operatora koje mogu pomoći u upravljanju stvorenim objektom odnosno videom. Stvaranje objekta klase *VideoCapture* ima tri inačice. Jedna kao argument prima konstantu tipa *int* koja zapravo predstavlja ID kamere, a takav pristup koristi se u sustavu za praćenje pozicije ljudskog tijela. Druga inačica prima argument tipa *string* koji predstavlja putanju do video datoteke koju se želi otvoriti. Nakon što je stvoren objekt tipa *VideoCapture,* potrebno je provjeriti je li on stvoren na pravi način i postoji li video datoteka, odnosno kamera s postojećim ID. Česta pogreška je pokušaj otvaranja video datoteke s formatom koji nije podržan. Osim videa, objekt pohranjuje informacije o rezoluciji, broju sličica i drugo. Pohranjivanje, odnosno stvaranje video datoteka slično je čitanju istih. U programskom kodu 4.5. prikazan je kod za čitanje videa s web kamere. U izvornom kodu nakon stvaranja objekta tipa *VideoCapture,* slijedi provjera otvaranja kamere, nakon čega se otvara petlja koja će očitavati sliku po sliku. To je ostvareno pomoću funkcije *.read()* koja kao argument prima objekt tipa *Mat.* Nakon uspješne provjere čitanja slike slijedi prikaz pomoću funkcije *imshow()* i čekanje korisnikovog unosa s tipkovnice pomoću funkcije *waitKey().*

Programski kod 4.5. Program za čitanje videa s web kamere

```
#include <opencv2\opencv2.hpp>
#include <iostream>
int main()
{
   cv::Mat slika;
   /* Otvaranje video ulaza odnosno kamere sa ID-em 0. */
   cv::VideoCapture video{ 0 };
   cv::namedWindow("Video", cv::WINDOW_AUTOSIZE);

   if (!video.isOpened())
   {
          std::cerr << "Nemogucnost otvaranja video izvora." << std::endl;
          return -1;
   }
   /* Petlja za ocitavanje i prikazivanje slika. */
   while (true)
   {
          if (!video.read(slika))
          {
                 std::cerr << "Problem sa očitavanjem slika" << std::endl;
                 break;
          }
          cv::imshow("Video", slika);
          if(cv::waitKey(1) == 27)
                 break;
   }
   return 0;}
```

4.5.3. Funkcija *namedWindow()*

Funkcija *namedWindow()* koristi se za stvaranje prozora na kojem se želi prikazati slika ili video. Funkcija je korisna jer pruža mogućnost odabira različitih funkcionalnosti prozora. Kao argumente prima objekt tipa *string* koji predstavlja ime i naslov prozora te konstantu tipa *int* koja označava kakvu će funkcionalnost imati prozor (hoće li biti preko cijelog zaslona, hoće li se moći skalirati prema slici i slično). U programskom kodu 4.5. navedena je linija koda koja definira prozor imena „video“ s konstantom *WINDOW_AUTOSIZE,* koja predstavlja prozor koji će se prilagoditi dimenzijama videa i neće se moći skalirati.

4.5.4. Funkcija *imshow()*

Funkcija *imshow()* nalazi se u nekoliko primjera programskog koda, a njena funkcija je prikaz slike ili videa na prozor. Prvi argument funkcije je tipa *string* koji predstavlja ime prozora na kojem će se prikazati. Drugi argument funkcije je izvor slike tipa *Mat*. Ako funkcija poziva ime prozora koji nije stvoren, ona će ga sama stvoriti.

4.5.5. Funkcija *waitKey()*

Funkcija *waitKey()* prima argument tipa *int.* Argument predstavlja vrijeme u milisekundama koje funkcija treba čekati na unos korisnika. Moguće je proslijediti i vrijednost nula, što zapravo znači da će funkcija čekati sve dok korisnik ne pritisne tipku. Kada je pritisnuta tipka na tipkovnici, funkcija vraća ASCII kod znaka, a ako je isteklo definirano vrijeme vraća -1.

4.6. Filtri

Kao osnova za prilagodbu i obradu slike potrebni su filtri. *OpenCV* posjeduje velik broj filtra različitih tipova i funkcija. U ovom poglavlju opisani su filtri koji su korišteni u sustavu za praćenje pozicije ljudskog tijela. U niže navedenim filtrima, opisani su samo ključni argumenti za ostvarivanje funkcionalnosti.

4.6.1. Funkcija *threshold()*

Funkcija *threshold()* ili funkcija binarnog ograničenja obično se koristi nakon nekoliko koraka obrade. Funkcija obrađuje piksel po piksel i uspoređuje ga sa zadanom granicom. Argumenti funkcije *threshold()* su: ulazna i izlazna slika tip *Mat,* granična vrijednost i maksimalna vrijednost tipa *double* te tip operacije *int.* Ako se pretpostavi da je cilj obraditi sliku tako da svi pikseli ispod vrijednosti 120 budu promijenjeni na 0, a svi iznad 120 na 255,

koristit će se funkcija *treshold()* s operatorm *THRESHOLD_BINARY*, graničnom vrijednošću 120 i maksimalnom vrijednošću 255.

4.6.2. Funkcija *morphologyEx()*

Funkcija *morphologyEx()* je generalna funkcija koja obuhvaća nekoliko različitih koraka u obradi bitovnih ili slika koje su u crno bijeloj izvedbi. Prije objašnjenja same funkcije potrebno je uvesti nekoliko novih pojmova koji su ključni za razumijevanje. Pri izvedbi operacija moguća je izvedba piksel po piksel kao što je slučaj s funkcijom *treshold()*. Međutim, ako je cilj operacije generalizirati određeno područje na slici, recimo uzeti apsolutnu vrijednost od određenog segmenta slike koju čini više piksela, onda se taj dio slike naziva jezgra (eng. *kernel*). Jezgra definira površinu, odnosno broj piksela nad kojim treba izvesti određenu operaciju. Uobičajene veličine jezgre su: 3x3, 5x5 i 7x7. Primjer izgleda jezgri prikazan je na slici 4.4. Centralni piksel u jezgri često se naziva točkom sidrenja (eng. *anchor point*).

1	1	1
1	**1**	1
1	1	1

1	1	1	1	1
1	1	1	1	1
1	1	**1**	1	1
1	1	1	1	1
1	1	1	1	1

Slika 4.4. Jezgra veličine 3x3 (lijevo) i 5x5 (desno)

Funkcija *morphologyEx()* kao argumente prima ulaznu i izlaznu sliku tipa *Mat,* konstantu tipa *int* koja označava operaciju, lokaciju centra tipa *Point* i broj iteracija tipa *int.* Korištene operacije bit će objašnjenje na primjeru sustava praćenja ljudskog tijela. Potrebno je naglasiti kako je ideja ove funkcije mogućnost izvođenja više različitih filtara i pristupa obradi slike jednom funkcijom. Sve operacije definirane za ovu funkciju temeljene su na filtrima *erode* i *dilate.* Filtar *dilate* je transformacija jezgre slike u njezin lokalni maksimum. Svi pikseli jezgre sa slike koju želimo obraditi postaviti će se na maksimalnu vrijednost koju sadrži ista jezgra izvorne slike. Filtar *erode* je suprotan pa će svi pikseli jezgre slike koju želimo obraditi biti

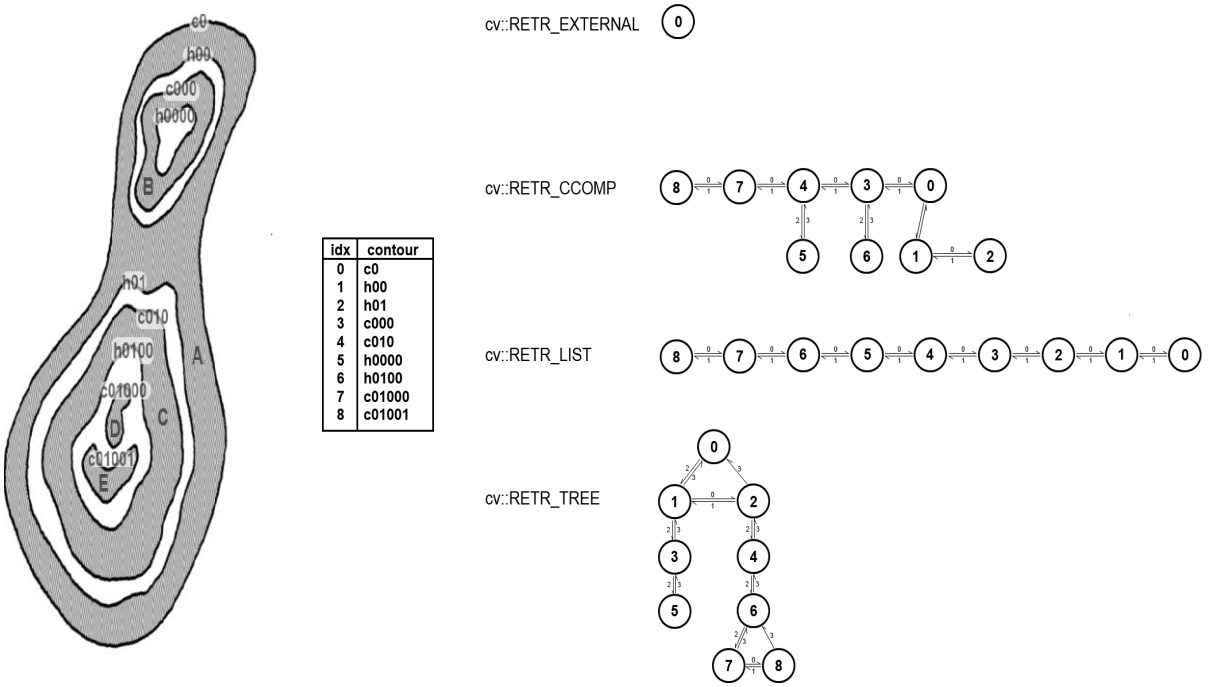
pretvoreni u minimalnu vrijednost jezgre s izvorne slike. To može biti od koristi kada je cilj rješavanje buke i smetnji te naglašavanje većih objekata [5, 6].

4.7. Konture

Nakon obrade slike izdvajanjem rubova i objekata, *OpenCV* nudi funkcije koje omogućavaju detekciju svih kontura na slici i obradu istih. Uz detekciju postoje i prilagođene funkcije za njihovo iscrtavanje. Ovdje je objašnjena osnovna funkcija za detekciju konture i osnovna funkcija za iscrtavanje konture.

4.7.1. Funkcija *findConturs()*

Funkcija *findConturs()* izvlači konture iz 8 bitne slike s jedim kanalom i pogodna je za izvlačenje objekata te njihovo prepoznavanje. Korištena je u sustavu za praćenje pozicije ljudskog tijela bez markera. Kao parametre prima ulaznu sliku, a kao izlaz daje *vektor* sačinjen od *vektora* tipa *Point* koji označava točke konture. Konstante tipa *int* definiraju način spremanja, odnosno pohranjivanja kontura. To je korisno kada je na slici više kontura koje čine jedna drugu pa ih je lakše sortirati i obrađivati. Na slici 4.5. moguće je da postoji više kontura unutar jedne konture. Prema tome, način spremanja kontura može biti lista ili stablo, ovisno o cilju. Konture se pritom pohranjuju kao kontura ili rupa, gdje kontura predstavlja vanjski obrub objekta, a rupa unutarnji. Druga konstanta predstavlja metodu spremanja, odnosno određuje koliko ključnih točaka konture će se pohraniti i na koji način.



Slika 4.5. Prikaz strukture kontura i način spremanja ovisno o modu [5]

4.7.2. Funkcija *drawConturs()*

Funkcija *drawConturs()* potpuno je prilagođena za obradu rezultata koje daje funkcija *findConturs()*. Zadatak funkcije *drawConturs()* je nacrtati dobivenu konturu uz nekoliko fleksibilnosti. Kao argumente prima sliku po kojoj crta konture te *vektor* od *vektora* tipa *Point* koji vraća funkcija za traženje kontura. Argument tipa *int* služi za odabir indeksa konture koja se želi crtati ili vrijednost -1 ako je potrebno nacrtati sve detektirane konture. Ostatak argumenta sličan je prijašnjim funkcijama za crtanje kao što su boja, debljina linije i slično.

4.8. Oduzimanje pozadine

Metoda oduzimanja pozadine (eng. *background subtraction)* popularna je u mnogim aplikacijama računalnog vida pa zato *i OpenCV* ima dva algoritma koja predstavljaju metodu oduzimanja pozadine. Metoda se temelji na usporedbi prošle pozadine s trenutnom i svaka promjena je zapravo novi objekt na sceni. Međutim, promjena osvjetljenja tijekom dana, vjetar koji giba grane, auti koji se kreću u pozadini, sve su to problemi koji utječu na kvalitetu metode. Osim rješavanja spomenutih problema pomoću filtara navedenih u poglavlju 4.6., dobar pristup rješenju je ažuriranje pozadine. Nakon određenog vremena objekt koji je došao na scenu, ako se dovoljno dugo zadrži i sam postaje dio pozadine. Ako se pretpostavi da se u vanjskim uvjetima

osvjetljenja sunce polako spušta, algoritam će imati dovoljno vremena da si prilagodi pozadinu kako pada sunce. Obje metode bit će detaljno objašnjene u primjeni sustava praćenja pozicije ljudskog tijela.

5. SUSTAV PRAĆENJA POZICIJE LJUDSKOG TIJELA

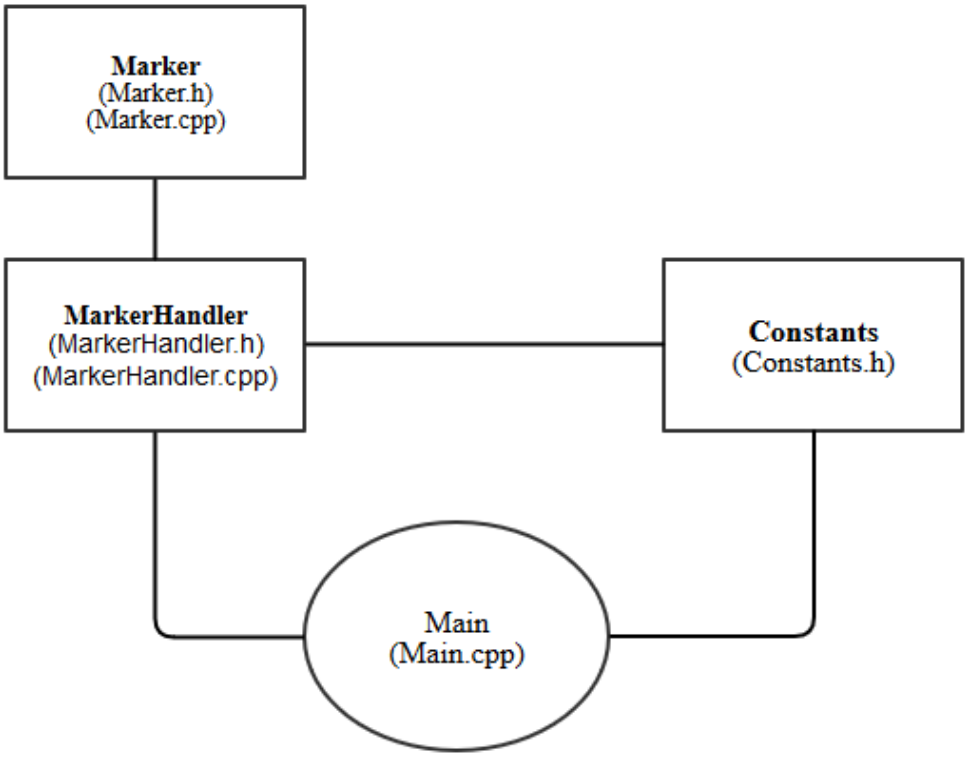
5.1. Opće informacije o praćenju pozicije ljudskog tijela

Usprkos razvoju koji traje više od 15 godina problem pozicije ljudskog tijela još nije riješen u potpunosti. Međutim, zahtjevi za rješenjem tog problema postoje sa svih aspekata ljudskog djelovanja, iako je iznimno težak. Razlog tome su varijabilne pojave koje je teško predvidjeti i na koje nije moguće utjecati, a to su: različita pojava ljudi na slici, različiti uvjeti osvijetljena, različita fizička građa ljudi, kompleksnost ljudskog kostura i fizionomije, spajanje i sudaranje ljudskih dijelova tijela na slici, gubitak informacije 3D objekta pri pretvorbi u 2D sliku i slično [3, 4]. Postoji velik broj pristupa rješavanju ovog problema, ali niti jedan od njih ne rješava gore navedene probleme i nije rješenje koje radi u većini uvjeta. To vrijedi i za dvije metode korištene u ovom radu. Prva koristi markere tipa *aruco*[6] koji su pričvršćeni na ljudsko tijelo i pomažu definirati ključne točke ljudskog tijela. Radi se o metodi koja funkcionira u 2D prostoru, odnosno nema informacije o dubini i udaljenosti. Druga metoda koristi se pristupom oduzimanja pozadine i pogodna je za sustave video nadzora i praćenja objekata u pokretu. Nedostatak ove metode je generirani šum na slici pa nije robusno primjenjiva. Detalji ovih pristupa navedeni su u narednim poglavljima.

5.2. Sustav praćenja pomoću *aruco* markera

Korištenje markera prilikom detekcije značajno olakšava samu detekciju, određivanje pozicije, određivanje dijela tijela i gibanje. U ovom radu upravo se operacije oko markera prožimaju kroz cijeli izvorni kod obzirom na to da markeri obavljaju gotovo svu funkcionalnost praćenja. Na slici 5.1. prikazana je struktura sustava koja se sastoji od klase *Marker, MarkerHandler,* datoteke zaglavlja *Constants* i datoteke *Main.* U narednim poglavljima bit će objašnjeni cijeli kod kao i međusobni odnosi datoteka. Prije samih datoteka i klasa bit će objašnjena funkcija koja se koristi za detekciju *aruco* markera.

[6] *Aruco* je modul koji spada u repozitorij opencv_contrib.



Slika 5.1. Struktura sustava za praćenje pozicije ljudskog tijela pomoću markera

5.2.1. Funkcija *detectMarkers()*

Funkcija *detectMarkers()* nalazi se u području imenika *cv::aruco::*. Njena glavna uloga je pronaći markere i izvući korisne informacije o istima. Kao parametre funkcija *detectMarkers()* prima sliku na kojoj trebaju biti markeri i objekt koji predstavlja tip imenika, odnosno skupinu markera koja se traži. Treći parametar je *vektor* od *vektora* tipa *Point* u koji se spremaju koordinate vrhova pronađenog markera počevši od gornjeg lijevog vrha pa u smjeru kazaljke na satu. Zadnji argument je *vektor* tipa int za ID svakog markera. Funkcija prima i argumente za odbačene kandidate i prilagođeno traženje, ali to nije potrebno za ovaj sustav.

5.2.2. Klasa *Marker*

Klasu *Marker* čine dvije datoteke, a to su *Marker.h* i *Marker.cpp*. Razlog tome je razdvajanje deklaracije od definicije što je preporučljivo zbog standarda pisanja koda te ako postoji potreba za pozivanjem klase te stvaranjem objekata u više datoteka programa. Ova klasa potpuno je prilagođena rezultatima funkcije *detectMarkers()* i služi za dodatnu obradu dobivenih informacija. U programskom kodu 5.1. prikazana je deklaracija klase *Marker*, a u izvornoj datoteci nalaze se i komentari kojih ovdje nema s obzirom na to da će kod biti objašnjen.

Programski kod 5.1. Deklaracija klase *Marker*

```
#pragma once
#ifndef MARKER_H
#endif MARKER_H

#include <opencv2\core.hpp>
#include <vector>

class Marker
{
public:

   Marker(int markerNum, std::vector<cv::Point2f> markerCorners, cv::Mat &frame);
   cv::Point2f getMarkerCenter();
   int getMarkerID() const;

 private:

    cv::Mat capFrame;
    int markerID;
    cv::Point2f center;
    std::vector<cv::Point2f> markerPoints;
    void calculateMarkerCenter();
};
```

Programski kod počinje s predprocesorskim naredbama *#pragma once, #ifndef MARKER_H i #endif MARKER_H* koje se koriste za zaštitu uključivanja i izbjegavanje višestrukih deklaracije klase. Nakon toga slijede naredbe *#include* koje omogućuju pristup objektima tipa vektor i svim objektima biblioteke *OpenCV*. Slijedi deklaracija klase *Marker*, a javno sučelje klase čine konstruktor s četiri parametra, funkcija za dohvaćanje centra i ID samog markera koji je stvoren. Radi se o funkciji koja ima kvalifikator *const*, a on označava da funkcija ne radi nikakve promjene podatkovnih članova. U privatnim članovima je funkcija koja se tiče unutarnje implementacije algoritma za izračun centra i podatkovni članovi gdje su pohranjene ključne informacije objekta. Konstruktor zadan s četiri parametra predstavlja povezanost s funkcijom za detekciju koja vraća iste argumente.

Ako se obrati pozornost na dizajn klase moguće je zaključiti da ona ima dobro definirano javno sučelje preko funkcija za dohvaćanje centra markera i njegovog ID broja, dok unutarnja implementacija u ovom trenutku nije bitna. Time je osiguran dobar nivo apstrakcije i nije potrebno znati na koji način funkcionira izračun centra i slični algoritmi unutar objekta. Ono što je bitno dobiti iz jednog objekta tipa *Marker* su njegov centar i njegov ID. Nakon što je javno sučelje kvalitetno definirano, ako postoji zanimanje za unutarnjom implementacijom klase ili

poboljšanjem implementacije, treba pogledati datoteku za definiciju *Marker.cpp* u programskom kodu 5.2.

Programski kod 5.2. Definicija klase *Marker*

```
#include "Marker.h"
#include <opencv2\imgproc.hpp>

Marker::Marker(int markerNum, std::vector<cv::Point2f> markerCorners, cv::Mat
&frame)
{
   markerID = markerNum;
   capFrame = frame;
   for (int i = 0; i < markerCorners.size(); i++)
   {
          markerPoints.push_back(markerCorners[i]);
   }
}

int Marker::getMarkerID() const
{
   return markerID;
}

void Marker::calculateMarkerCenter()
{
   cv::LineIterator iter(capFrame, markerPoints[0], markerPoints[2], 8, false);
   for (int i = 0; i < iter.count; i++)
   {
          if (iter.count / 2 == i)
          {
                 center = iter.pos();
          }
          ++iter;
   }
}

cv::Point2f Marker::getMarkerCenter()
{
   Marker::calculateMarkerCenter();
   return center;
}
```

Definicija klase počinje uključivanjem datoteke zaglavlja same klase i modula *OpenCV imgproc* koji sadrži *LineIterator* potreban za pronalazak centra markera. Nakon zaglavlja slijedi definicija konstruktora koji pohranjuje vrijednosti u lokalne varijable objekta. Slijedi funkcija *getMarkerID()* koja čini javno sučelje klase i vraća ID *Markera.* Nakon toga, funkcija privatnog prava pristupa *calculateMarkerCenter()* koja sadrži algoritam za izračun centra markera. Ona se koristi iteratorom *LineIterator* i koordinatama vrhova markera posloženih redom u smjeru kazaljke na satu. Iteratoru su proslijeđeni vrhovi dijagonala markera koji osiguravaju kretanje po dijagonali. Čim se pronađe polovica dijagonale, pronađen je i centar markera koji je pohranjen u

varijablu privatnog prava pristupa. Na kraju dolazi funkcija *getMarkerCenter()* koja čini javno sučelje objekta te poziva funkciju za izračun centra markera i vraća isti.

5.2.3. Datoteka zaglavlja *Constants.h*

Kako ne bi dolazilo do zagađivanja globalnog područja jer sustav zahtjeva velik broj konstanti, definirano je zaglavlje u području imenika *constants*. U programskom kodu 5.2. prikazane su sve definirane konstante korištene u sustavu.

Programski kod 5.3. Definicija datotke *Constants.h*

```
#pragma once
#ifdef CONSTANTS_H
#endif CONSTANTS_H

#include <string>

namespace constants
{
   const int END_KEY = 30;
   const int VIDEO_SOURCE = 0;
   const int MIDDLE_MARKER_CHEST = 3;
   const int MIDDLE_MARKER_BELLY = 10;
   const int UPPER_BODY_LIMIT = 7;
   const int LOWER_BODY_LIMIT = 25;
   const int DISTANCE_REF_MARKER_ONE = 29;
   const int DISTANCE_REF_MARKER_TWO = 30;
   const int REFRENCE_DISTANCE = 46;
   const int MARKER_MEASURE_START = 3;
   const int MARKER_MEASURE_END = 10;
   const std::string WINDOW_NAME{ "Vision system" };
}
```

Programski kod počinje predprocesorskim naredbama i uključivanjem *string* klase, nakon čega slijedi područje imenika *constants.* Prednost ovakvog pristupa je u tome što su sve konstante dodatno sakrivene u imenik i imaju kvalifikator *const* koji označava konstantu. Kvalifikator onemogućava novu definiciju varijable, odnosno promjenu vrijednosti, jer prilikom izgradnje koda kompajler javlja pogrešku. Osim navedenih prednosti, glavna prednost ovakvog pristupa upravljanju konstantama je što se nalaze na jednom mjestu pa je moguće upravljati cijelim programom iz jedne datoteke. Na primjer, ako projekt ima 100 datoteka izvornog koda i jedna konstanta je raspršena kroz svih 100 datoteka kao čisti broj, za mijenjanje konstante potrebno je proći sve datoteke i zamijeniti ih ručno. Kod pristupa iz programskog koda 5.3. potrebno je promijeniti konstantu samo na jednom mjestu.

5.2.4. Klasa *MarkerHandler*

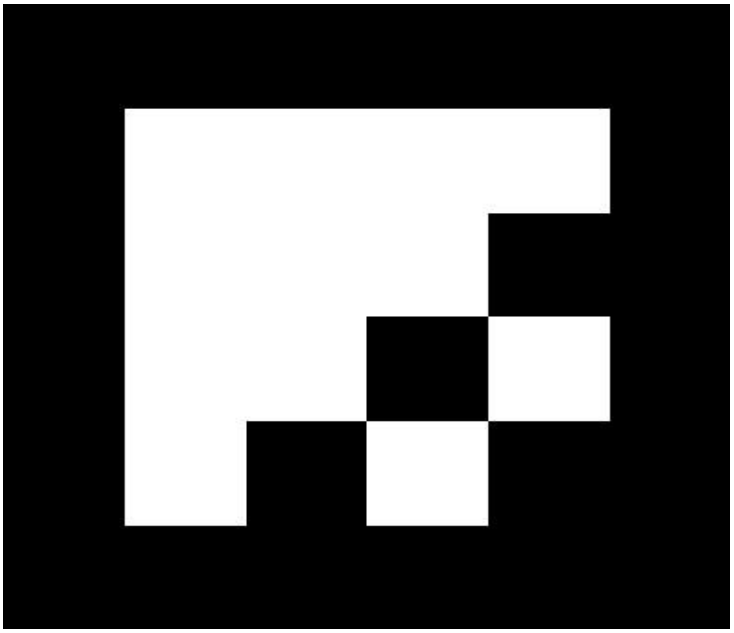
Uloga klase *MarkerHandler* je pronaći sve *aruco* markere i biti pristupna točka za sve pronađene markere, odnosno objekte klase *Marker*. To znači da mora izvoditi operacije kao što su detekcija, crtanje i mjerenje na većem broju markera. Klasu čine dvije datoteke, *MarkerHandler.cpp* i *MarkerHandler.h*. Programski kod 5.4. obuhvaća deklaraciju klase u datoteci zaglavlja.

Programski kod 5.4. deklaracija klase MarkerHandler

```
#pragma once
#ifdef MARKERHANDLER_h
#endif MARKERHANDLER_h

#include "Marker.h"
#include <opencv2\aruco.hpp>
#include <vector>

class MarkerHandler
{

public:

   MarkerHandler(cv::Mat &frame) : refrenceFrame(frame) {};
   void markerTracking();
   void showMarkerDistance();

private:

   void detectMarkers(cv::Mat& frame, cv::Ptr<cv::aruco::Dictionary> dictionary,
                      std::vector<Marker>& detectedMarkers);

   void connectDetectedMarkers(cv::Mat& frame, std::vector<Marker>
                               detectedMarkers);

   cv::Mat& refrenceFrame;
   cv::Ptr<cv::aruco::Dictionary> markerDictionary=cv::aruco::
   getPredefinedDictionary(cv::aruco::PREDEFINED_DICTIONARY_NAME::DICT_4X4_50);
   std::vector<Marker> detectedMarkers;
};
```

Nakon pretprocesorskih naredbi za zaštitu od uključivanja, slijede naredbe *#include* koje obuhvaćaju klase *Marker, Vektor i* datoteku zaglavlja *aruco.hpp*. Klasa deklarira konstruktor koji prima jedan argument tipa *Mat* koji treba predstavljati trenutnu sliku iz video izvora, nakon čega slijede dvije funkcije tipa *void* koje čine javno sučelje ove klase. Funkcije s privatnim pravom pristupa čine i obavljaju glavne zadatke ove klase, a to su detekcija i spajanje objekata klase *Marker*. Podatkovni članovi klase su referenca na sliku, pokazivač (eng. *pointer*) i *vektor* koji pohranjuje objekte tipa *Marker*. Tip *cv::Ptr* je tip pokazivača biblioteke *OpenCV*, a u ovom slučaju, pod nazivom *markerDictionary*, predstavlja tip markera koje je potrebno pronaći na

slici. Tip markera nalazi se pod imenikom *cv::aruco::Dictionary*, dok se za pristup vrijednosti koristi funkcija s argumentom koji pokazuje na jednu od biblioteka markera. U ovom slučaju radi se o argumentu *cv::aruco::PREDEFINED_DICTIONARY_NAME::DICT_4X4_50*. Tip markera zapravo označava na koji način su markeri napravljeni i koliko ih ima. U ovom primjeru radi se o markeru s četiri horizontalna i vertikalna bloka koji mogu sadržavati jedinicu ili nulu, odnosno crnu ili bijelu boju. Količina markera u biblioteci na koju je inicijaliziran pokazivač je 50. Na slici 5.2. prikazan je primjer *aruco* markera iz biblioteke *DICT_4X4_50*, čiji ID iznosi 8.



Slika 5.2. *Aruco* marker

Same markere potrebno je stvoriti, za što datoteka zaglavlja *aruco.hpp* ima prigodne funkcije. Tako su i za sustav praćenja pozicije ljudskog tijela generirani markeri koji su kasnije printani na papir.

Nakon deklaracije slijedi i definicija klase koja se nalazi u datoteci *MarkerHandler.cpp*. U narednom primjeru prikazanom programskim kodom 5.5 izostavljeni su komentari. Izvorni kod počinje s naredbama *#include* koja obuhvaća sve potrebne datoteke zaglavlja. Nakon toga slijedi definicija funkcije *detectMarkers()* koja se nalazi u programskom kodu 5.5.

Programski kod 5.5. Definicija funkcije *detectMarkers()*

```
void MarkerHandler::detectMarkers(cv::Mat& frame, cv::Ptr<cv::aruco::Dictionary>
dictionary, std::vector<Marker>& detectedMarkers)
{
   std::vector<std::vector<cv::Point2f>> markerCorners;
   std::vector<int> markersID;

   cv::aruco::detectMarkers(frame, dictionary, markerCorners, markersID);
   cv::aruco::drawDetectedMarkers(frame, markerCorners, markersID);

   if (markersID.size() > 0)
   {
          for (int i = 0; i < markersID.size(); i++)
          {
                 detectedMarkers.push_back(Marker
                                  { markersID[i], markerCorners[i], frame });
          }
          auto ReOrder = [](Marker one, Marker two)
                        { return(one.getMarkerID() < two.getMarkerID()); };
          sort(detectedMarkers.begin(), detectedMarkers.end(), ReOrder);
   }
}
```

Funkcija prima trenutnu sliku s videa tipa *Mat,* pokazivač na spomenutu biblioteku markera i referencu na *vektor* koji pohranjuje objekte tipa *Marker*. Bitno je naglasiti da se radi o referenci *vektora* jer se niže u programskom kodu isti vektor puni s objektima i izvode se operacije nad objektima. Iz tog razloga koristi se referenca za upravljanje izvornim vektorom, a ne preslikama. Slijedi definicija lokalnih varijabli za pohranjivanje vrhova i ID markera. Te varijable potrebne su za funkciju detekcije markera. Nakon toga slijedi funkcija *cv::aruco::drawDetectedMarkers* koja označava pronađene markere i na gornji lijevi vrh stavlja crveni kvadratić te u centar stavlja ID samog markera. Nakon što su obavljene funkcije detekcije i crtanja, moguće je izvoditi operacije na istima. U *for* petlji stvaraju se objekti tipa *Marker* s proslijeđenim parametrima dobivenih iz detekcije. Nakon petlje slijedi definicija lambda izraza. Lambde izrazi predstavljaju bezimene funkcije koje se ponašaju kao funkcijski objekti [7]. Definiran je lambda izraz koji prima dva markera i vraća *bool* vrijednost, ovisno o tome čiji ID je veći. Potrebno je primijetiti kvalifikator *auto* kojime se naznačava kompajleru da sam odluči o tome kojega tipa će biti povratna vrijednost lambde. Slijedi funkcija *sort()* koja kao parametre prima iteratore na početak i na kraj niza, funkcijski objekt ili lambda izraz koji treba vratiti *bool* vrijednost. Njezin zadatak je sortirati markere po njihovoj vrijednosti, a ona je definirana u biblioteci standardnih predložaka programskog jezika C++.

Programski kod 5.6. Definicija funkcije *connectDetectedMarkers()*

```
void MarkerHandler::connectDetectedMarkers(cv::Mat& frame, std::vector<Marker>
detectedMarkers)
{
          std::vector<cv::Point2i> centerPointsLowerBody;
          std::vector<cv::Point2i> centerPointsUpperBody;
          std::vector<cv::Point2i> middlePoints;
          for (auto object : detectedMarkers)
          {

                 if (object.getMarkerID() < constants::UPPER_BODY_LIMIT)
                 {
                        centerPointsUpperBody.push_back(object.getMarkerCenter());
                     if(object.getMarkerID() == constants::MIDDLE_MARKER_CHEST)
                              middlePoints.push_back(object.getMarkerCenter());
                 }
                 else
                 {
                     if(object.getMarkerID() < constants::LOWER_BODY_LIMIT)
                        {
                        centerPointsLowerBody.push_back(object.getMarkerCenter());
                         if(object.getMarkerID()== constants::MIDDLE_MARKER_BELLY)
                              middlePoints.push_back(object.getMarkerCenter());
                        }
                      else
                        continue;
                 }

          }

   cv::polylines(frame, centerPointsUpperBody, false, cv::Scalar{ 0,0,0 }, 3, 8);
   cv::polylines(frame, centerPointsLowerBody, false, cv::Scalar{ 0,0,0 }, 3, 8);

   if (middlePoints.size() == 2)
   cv::line(frame, middlePoints[0], middlePoints[1],cv::Scalar{ 0,0,0 }, 3, 8);
   }
```

U programskom kodu 5.6. prikazana je definicija funkcije *connectDetectedMarkers()* koja kao argumente prima sliku po kojoj treba crtati i *vektor* koji sadrži objekte tipa *Marker*. Lokalni podatkovni članovi koriste se za pohranjivanje i razvrstavanje centralnih točaka markera po skupinama na točke gornjeg djela tijela, doljnjeg i centralne točke. Taj proces pohranjivanja i razvrstavanja odvija se u *for* petlji. Nakon što se točke nađu u željenim spremnicima, koriste se funkcije za crtanje linija *polylines() i line()* koje spajaju proslijeđene točke. Iz navedenog moguće je zaključiti da je zadatak ove funkcije razvrstati marker, odnosno njihove točke i spojiti ih. Programski kod 5.7. obuhvaća definiciju funkcije *showMarkerDistance()* čiji je zadatak pomoću referentnih markera izračunati i prikazati udaljenost između željenih markera u centimetrima.

Programski kod 5.7. Definicija funkcije *showMarkerDistance()*

```
void MarkerHandler::showMarkerDistance()
{
   cv::Point2f markerCenterOne;
   cv::Point2f markerCenterTwo;
   cv::Point2f markerChest;
   cv::Point2f markerBelly;
   for (auto object : detectedMarkers)
   {
          if (object.getMarkerID() == constants::DISTANCE_REF_MARKER_ONE)
                 markerCenterOne = object.getMarkerCenter();
          else if (object.getMarkerID() == constants::DISTANCE_REF_MARKER_TWO)
                 markerCenterTwo = object.getMarkerCenter();
          else if (object.getMarkerID() == constants::MARKER_MEASURE_START)
                 markerChest = object.getMarkerCenter();
          else if (object.getMarkerID() == constants::MARKER_MEASURE_END)
                 markerBelly = object.getMarkerCenter();
          else
                 continue;
   }
   if (markerChest.x > 0 && markerBelly.x > 0 && markerCenterOne. x > 0 &&
          markerCenterTwo.x > 0 )
   {
          cv::LineIterator iterator(refrenceFrame, markerCenterOne,
                                                 markerCenterTwo, 8, true);
          double numberOfPixels = iterator.count;
          cv::LineIterator iteratorTwo(refrenceFrame, markerChest, markerBelly,
                                                                 8,true);
          double numberOfPixelsChest = iteratorTwo.count;
          double distance = (constants::REFRENCE_DISTANCE / numberOfPixels) *
                                                        numberOfPixelsChest;

          std::string distanceInCm{ "Lenght chest: " };
          std::string temp = std::to_string(distance);
          distanceInCm.append(temp);
          distanceInCm.append(" cm");
          cv::putText(refrenceFrame, distanceInCm, cv::Point(400, 400),
          cv::HersheyFonts::FONT_HERSHEY_PLAIN, 1, cv::Scalar{ 0,0,0 }, 2, 8);
   }
}
```

Programski kod 5.7. počinje stvaranjem lokalnih varijabli tipa *Point.* One predstavljaju ključne točke markera koji će se mjeriti i biti referentni. Slijedi *for* petlja koja služi za pohranjivanje i razvrstavanje kao i u programskom kodu 5.6. Uvjetno izvođenje pomoću uvjetne naredbe *if* služi za kontrolu pronalaska traženih markera. Nakon toga se koristi *LineIterator* da bi se dobio broj piksela pomoću kojeg se izračunaju udaljenosti između traženih markera. Nakon što je izračunata vrijednost udaljenosti tipa *double*, ona se pretvara u tip *string* i ispisuje na sliku pomoću funkcije za tekst. Funkcija *markerTracking()* iz deklaracije klase *Markerhandler* obuhvaća funkcije *connectDetectedMarkers()* i *detectMarkers()* u jednu te nema drugu zadaću.

5.2.5. Datoteka *Main.cpp*

Kao što i samo ime predstavlja, radi se o datoteci koja obuhvaća funkciju *main()* odnosno glavnu funkciju programa i sustava za praćenje pozicije ljudskog tijela. Sva funkcionalnost koja obuhvaća pokretanje kamere i čitanje slika nalazi se u zasebnoj funkciji *visionSystem* koja je pozvana u glavnoj funkciji. U programskom kodu 5.8 nalazi se definicija funkcija *main()* i *visionSystem().*

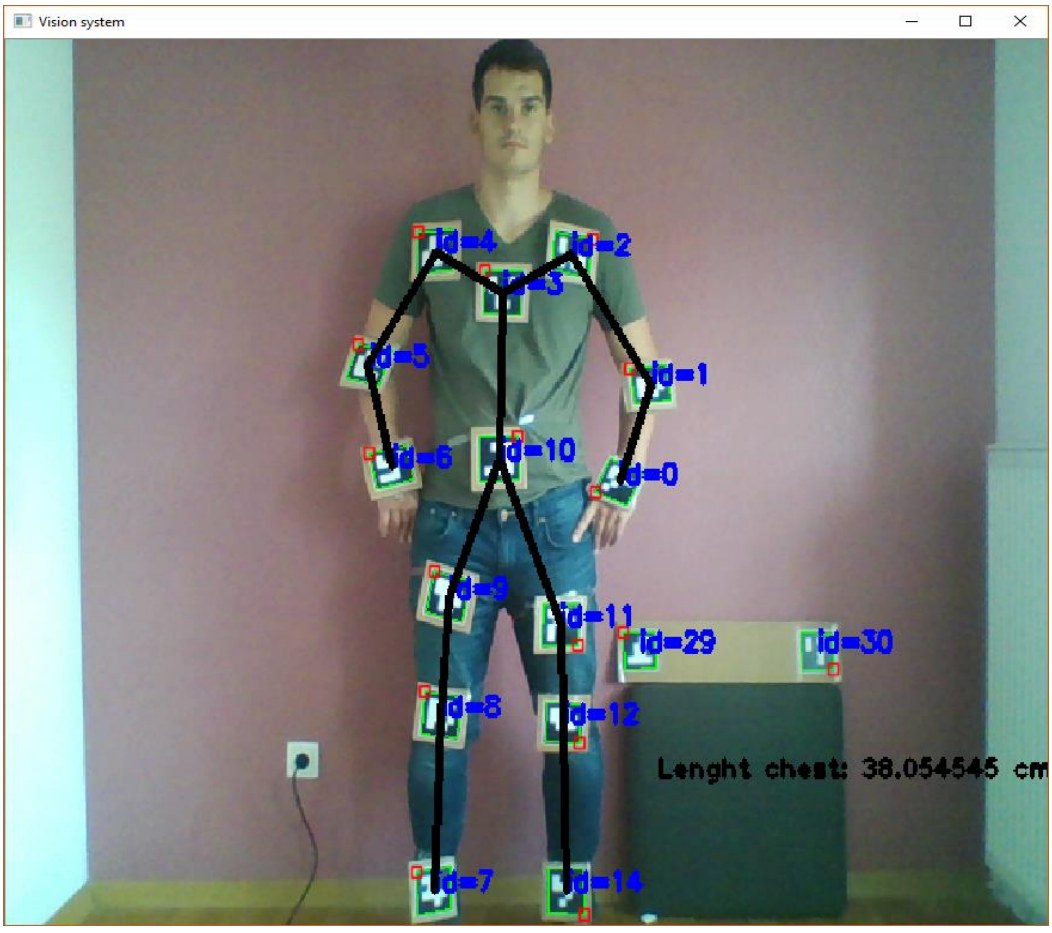
Programski kod 5.8. Definicija funkcije *main() i visionSystem()*

```
int visionSystem();

int main()
{
   visionSystem();
   return 0;
}
int  visionSystem()
{
   cv::Mat frame;
   cv::VideoCapture vid(constants::VIDEO_SOURCE);
   if (!vid.isOpened())
   {
          std::cout << "Problem with video source, check camera ID!" << std::endl;
          return -1;
   }
   cv::namedWindow(constants::WINDOW_NAME, cv::WINDOW_FREERATIO);
   while (true)
   {
          if (!vid.read(frame))
          {
                 std::cout << "Problem with loading frame!" << std::endl;
                 break;
          }
          MarkerHandler markerHandlerobject{ frame };
          markerHandlerobject.markerTracking();
          markerHandlerobject.showMarkerDistance();

          cv::imshow("Vision system", frame);
          if (cv::waitKey(constants::END_KEY) >= 0)
                 break;
   }
   return 0;
}
```

Kod počinje potpisom funkcije *visionSystem()* i pozivom iste unutar funkcije *main().* Funkcija *visionSystem()* ima zadatak dohvatiti video s web kamere, a zatim stvoriti objekt tipa *MarkerHandler* koji kao argument prima trenutnu sliku i poziv funkcija za praćenje te prikaz udaljenosti između pronađenih markera.

5.3. Rezultati sustava praćenja pomoću *aruco* markera

Praćenje ljudskog tijela pomoću *aruco* markera je u tehničkoj izvedbi zapravo praćenje samih markera koji su postavljeni na ključne točke osobe koju je cilj pratiti. Sustav funkcionira dok su markeri pronađeni, odnosno dok su dovoljno kvalitetno vidljivi samoj kameri. Ako se radi o kameri slabije kvalitete kao što su web kamere, nakon što se marker udalji tri metra od kamere, zbog niske rezolucije, kamera više ne prepoznaje marker. Dok su markeri dovoljno vidljivi, udaljenost koju je potrebno mjeriti između određenih ključnih točaka, daje zadovoljavajuće rezultate. Međutim, za mjerenje udaljenosti potrebno je imati referentnu dužinu ili predmet na samoj slici kako bi postojala referenca. U ovom slučaju radi se o dva referentna markera fiksne udaljenosti od 46 centimetra. Na slici 5.3. prikazani su rezultati praćenja i mjerenja pomoću *aruco* markera, dok se u prilogu nalaze dodatne slike.



Slika 5.3. Rezultat praćenja pomoću aruco markera

5.4. Sustav praćenja pomoću oduzimanja pozadine

U ovom pristupu koristi se praćenje čovjeka bez markera temeljeno na algoritmu oduzimanja pozadine. Pretpostavka je da kamera koja miruje uvijek gleda istu pozadinu. Kada se na pozadini pojavi novi objekt ili čovjek, algoritam ga izdvoji. *OpenCV* ima dva takva algoritma koji su detaljno objašnjeni u izvorima [3, 4]. Algoritam *BackgroundSubstractorKNN* bazira se na metodi strojnog učenja i klasifikaciji najbližeg susjeda (eng. *nearest neighbor classifier)* dok se algoritam *BacgroundSubstractorMOG2* bazira na Gaussovoj razdiobi. U narednim poglavljima bit će objašnjeni ključni koraci ovog sustava, dok se kod u cijelosti nalazi u prilogu.

5.4.1. Stvaranje objekata i primjenjivanje algoritma za oduzimanje pozadine

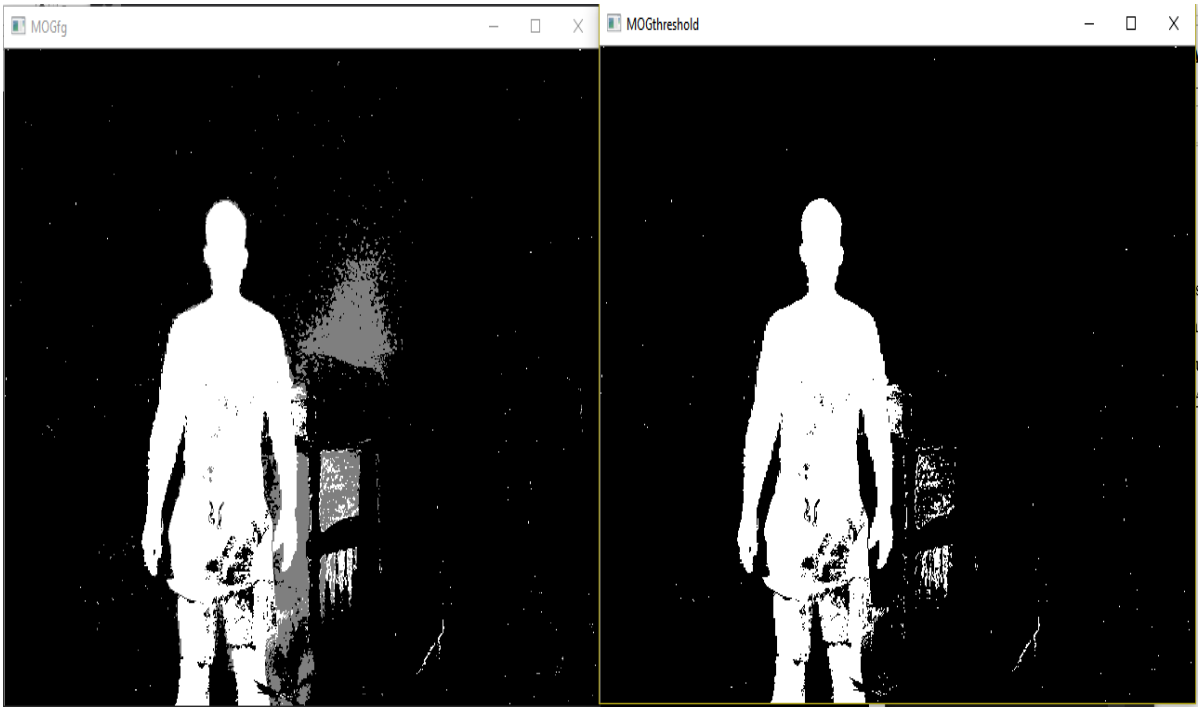
Kako bi se algoritmi mogli koristiti potrebno je stvoriti pokazivače na iste te ih primijeniti na dobivenu sliku. Programski kod 5.9 prikazuje primjenu algoritama za oduzimanje površine. Nakon deklaracije varijabli tipa *cv::Mat* koji se koriste u algoritmima, stvaraju se dva pokazivača na objekte tipa *BackgroundSubstractor* nakon čega se istima dodjeljuje vrijednost. Pomoću funkcija *createBackgroundSubstractor()* stvara se objekt s određenim parametrima. Kod KNN i MOG2 metoda prvi parametar predstavlja broj slika koji je potreban da algoritam potpuno osvježi pozadinu. Drugi parametar predstavlja udaljenost, odnosno granicu u kojoj piksel pripada ili ne pripada objektu. Obzirom na to da obje funkcije mogu pronaći sjene, parametar tipa *bool* određuje hoće li algoritam te sjene označiti ili ne.

Programski kod 5.9. Primjena algoritama za oduzimanje površine

```
cv::Mat frame;
cv::Mat fgMOG2;
cv::Mat fgKNN;

cv::Ptr<cv::BackgroundSubtractor> MOG2;
cv::Ptr< cv::BackgroundSubtractor> KNN;

MOG2 = cv::createBackgroundSubtractorMOG2(2000, 16,true);
KNN = cv::createBackgroundSubtractorKNN(2000,400, true);

/* Neki kod između. */
MOG2->apply(frame, fgMOG2);
KNN->apply(frame, fgKNN);

cv::threshold(fgMOG2, thresMOG2, 128, 255, cv::THRESH_BINARY);
cv::threshold(fgKNN, thresKNN, 128, 255, cv::THRESH_BINARY);
```

Nakon što je algoritmu naznačeno da označi sjene, on će postaviti piksele koji predstavljaju sjenu na vrijednost 127. Budući da je sjenu i ostale smetnje potrebno ukloniti, primjenjuje se

funkcija *threshold()* s graničnom vrijednosti 128. Na slici 5.4., s lijeve strane, nalazi se prikaz detektirane sjene vrijednosti 127, odnosno sive boje. Na desnoj strani slike je prikaz nakon uklanjanja sjene. Iako je sjena uklonjena, vidi se da je prisutan šum na slici.



Slika 5.4. Uklanjanje sjena sa slike

5.4.2. Dodatna obrada rezultata oduzimanja pozadine primjenom filtra

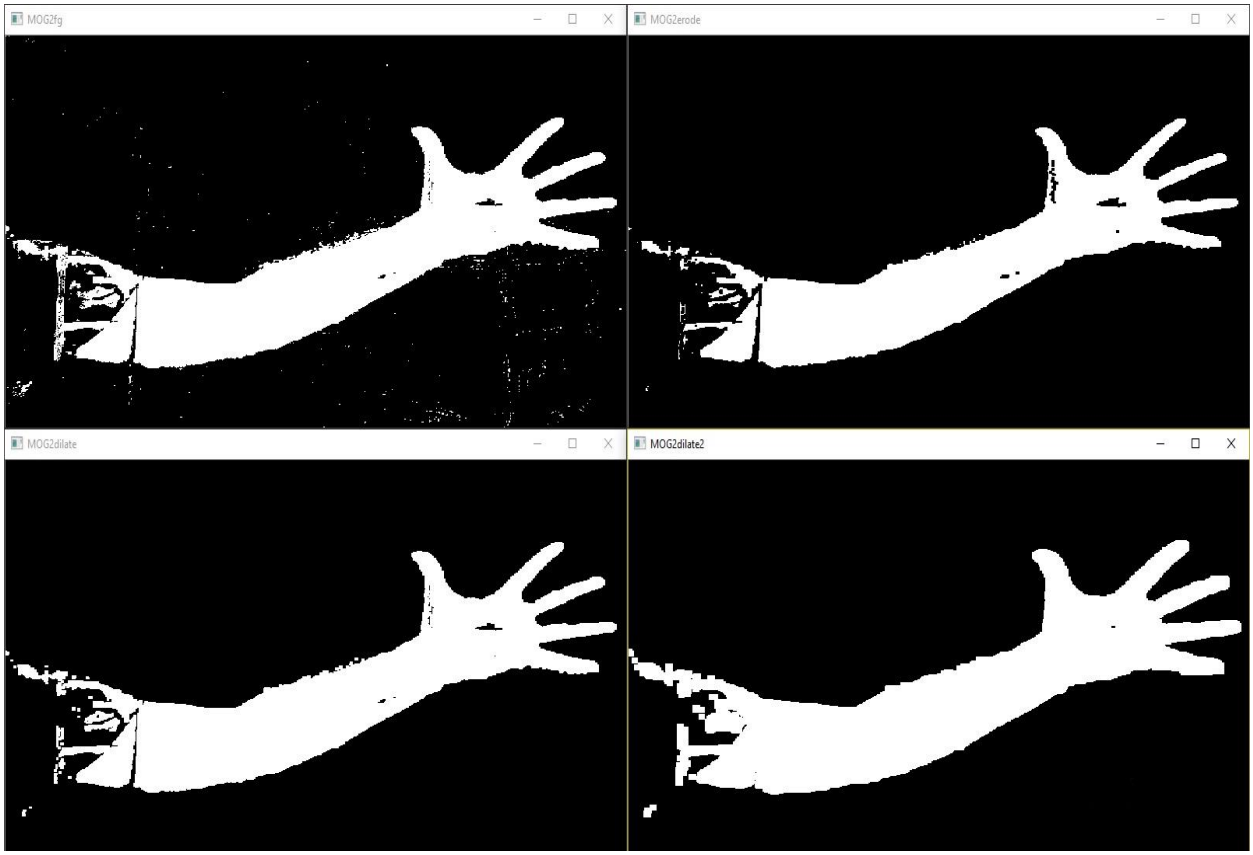
S obzirom na to da rezultati postupka oduzimanja pozadine često generiraju šum zbog promjene osvijetljena, blagih gibanja ili nedovoljnog kontrasta novog objekta, potrebno je dodatno obraditi rezultat, odnosno ukloniti šum određenim filtrima. Programski kod 5.10 prikazuje takve postupke obrade koji otklanjaju sve nedostatke nastale šumom.

Programski kod 5.10. Korištenje filtra *erode i dilate*

```
cv::Mat elementOpen = cv::getStructuringElement(cv::MorphShapes::MORPH_RECT,
cv::Size(3,3));
cv::Mat elementDilate = cv::getStructuringElement(cv::MorphShapes::MORPH_RECT,
cv::Size(5,5));

cv::Mat openMOG2, openKNN;
cv::morphologyEx(thresMOG2, openMOG2, cv::MORPH_OPEN, elementOpen,cv::Point(-1,-
1),1);
   cv::morphologyEx(thresKNN, openKNN, cv::MORPH_OPEN, elementOpen, cv::Point(-1,
-1), 1);
cv::Mat dilateMOG2, dilateKNN;
cv::dilate(openMOG2, dilateMOG2,elementDilate);
cv::dilate(openKNN, dilateKNN,elementDilate);
```

Programski kod 5.10. kod počinje strukturiranjem elemenata za obradu. O tome je bilo govora u poglavlju 4.6. koje se bavi filtrima. Strukturni element definira na kakvoj i kojoj površini će se neki filtar primijeniti. Parametri koje prima su oblik izdvojenog segmenta slike (može biti u obliku pravokutnika, kruga ili kvadrata), veličina jezgre i centralna točka. Nakon pripreme argumenata slijedi poziv funkcije *morphologyEx()* koja se koristi za filtriranje, a koja je opisana u poglavlju 4.6.2.

U ovom sustavu korišteni su filtri *erode i dilate* koji čine osnovne filtre za bitovne slike. *Dilate* je postupak u kojem se dana jezgra pretvara u svoj lokalni maksimum. To znači da će mjesta visoke vrijednosti, koja su razdvojena malom udaljenosti biti stopljena u jednu cjelinu. Filtar *erode* radi upravo suprotno i pretvara jezgru u njezin lokalni minimum što rezultira odvajanjem objekata u manje segmente i potpunim uklanjanjem malih dijelova. U programskom kodu 5.10. moguće je primijetiti da se prije filtra *dilate* poziva i filtar *open* koji zapravo obuhvaća filtre *erode* i *dilate.*



Slika 5.5. Izvorni rezultat oduzimanja pozadine s uklonjenom sijenom i postupna primjena filtara *erode i dilate*

Na slici 5.5. nalazi se postupak obrade slike koji je korišten u sustavu praćenja pomoću oduzimanja pozadine na testnom uzorku ruke. U gornjem lijevom dijelu slike nalazi se izvorna slika nakon oduzimanja pozadine i uklanjanja sijena, u gornjem desnom dijelu slijedi prva primjena filtra *erode* koji uklanja sve smetnje oko ruke, nakon čega slijedi dvostruka primjena filtra *dilate*. Moguće je primijetiti kako se ruka najčišće vidi na donjem desnom dijelu slike.

5.4.3. Izdvajanje i iscrtavanje konture

Nakon što su rezultati oduzimanja pozadine dodatno obrađeni i prilagođeni kako bi se dobio traženi oblik, moguće je početi tražiti konturu. Programski kod 5.11. obuhvaća traženje konture, izdvajanje najveće konture i iscrtavanje iste. Kod počinje funkcijom *findConturs()* koja u ovom slučaju izdvaja samo vanjske konture i pohranjuje ih u običan niz. Nakon toga počinje *for* petlja čiji je cilj izdvojiti najveću konturu koja se pojavila na sceni, što je u ovom slučaju čovjek. Nakon što je pronađena najveća kontura ista se i ocrtava na izvornoj slici pomoću funkcije *drawConturs* tako što joj se prosljeđuje ID najveće pronađene konture.

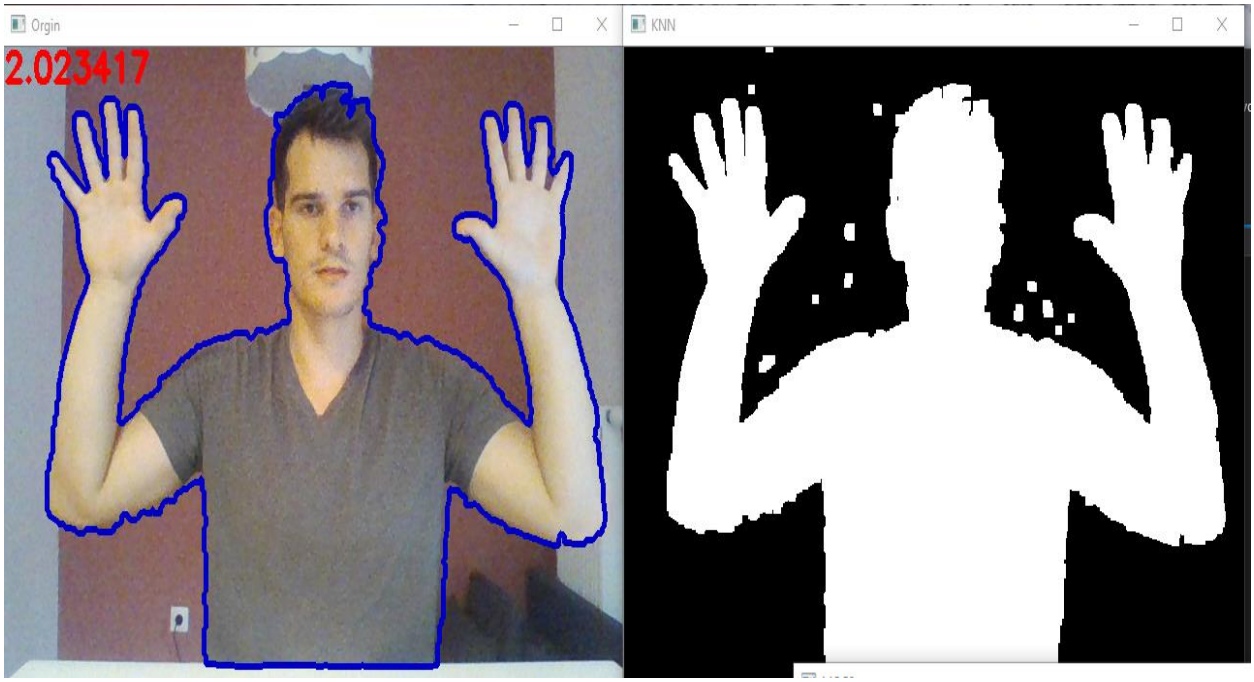
Programski kod 5.11. Izdvajanje i iscrtavanje konture

```
std::vector<std::vector<cv::Point>> conturs;
cv::findContours(dilateKNN, conturs, cv::RETR_EXTERNAL, cv::CHAIN_APPROX_NONE);

int largestArea = 0;
int conturIndex = 0;
   for (int i = 0; i < conturs.size(); i++)
   {
          double area = cv::contourArea(conturs[i]);
          if (area > largestArea)
          {
                 largestArea = area;
                 conturIndex = i;
          }
   }
cv::drawContours(frame, conturs, conturIndex, cv::Scalar{ 200,0,0 }, 3, 8);
```

5.4.4. Rezultati sustava za izdvajanje pozadine

Ovaj pristup praćenju čovjeka ima nekoliko zahtjeva da bi funkcionirao na željeni način. Osvjetljenje mora biti stabilno, kamera mora biti fiksna i poželjno je da postoji razlika u kontrastu između objekta kojega želimo pratiti i pozadine s obzirom na to da se radi o metodi čistog uspoređivanja piksela. Ovaj sustav napravljen je da izdvoji najveći objekt na sceni, a za izdvajanje čovjeka potrebna je dodatna obrada. Iako u pristupu s markerima mjerenje udaljenosti nije zahtjevan zadatak, u ovom pristupu je izuzetno zahtjevno mjeriti duljinu udova i rezultati

nisu dovoljno precizni. U pristupu s markerima rezultat je točan unutar jednog centimetra pogreške. Na slici 5.6. nalazi se rezultat sustava za oduzimanje pozadine, dok se u prilogu nalaze dodatne slike.



Slika 5.6. Rezultat sustava oduzimanja pozadine.

6. ZAKLJUČAK

Računalni vid kao grana računarstva u stalnom je razvoju. Primjena računalnog vida u budućnosti bi mogla igrati veliku ulogu u sustavima interneta stvari. Ključnu ulogu u računalnom vidu ima obrada slike u digitalnom formatu. Ozbiljna obrada slike počinje s kvalitetnom slikom, odnosno ulaznim signalom koji ima dovoljno informacija za obradu što je pokazao i ovaj rad u kojem je korištena web kamera. Zbog loše kvalitete kamere rezultati algoritama nisu dovoljno dobri, ali pokazuju potencijal. Kada se govori o kvalitetnim algoritmima, biblioteka *OpenCV* ima njih nekoliko stotina koji su potpuno optimizirani i trenutno predstavljaju gotovo pa najbolje rješenje većine problema računalnog vida. Budućnost biblioteke je dodavanje novih algoritama kako i računalni vid kao grana računarstva bude rasla. Sukladno tome znanje korištenja biblioteke *OpenCV* u budućnosti bi programerima moglo biti iznimno korisna jer su sustavi za praćenje poze ljudskog tijela bazirani kompletno na biblioteci *OpenCV*.

Razvijeni sustavi praćenja pozicije ljudskog tijela predstavljaju specijalizirane pristupe u rješavanju problema koji su primjenjivi samo u određenim uvjetima. To znači da je potrebno prilagoditi osvjetljenje, fiksirati kameru i postaviti markere da bi sustavi bili funkcionalni i davali zadovoljavajuće rezultate. Prvi pristup rješavanju problema koristi markere. Ovaj pristup kao bazu iz biblioteke koristi modul *aruco* iz kojega su i generirani markeri. Sustav pomoću markera daje zadovoljavajuće rezultate praćenja i mjerenja ključnih točaka ljudskog tijela. Nedostatak ovoga pristupa je što zahtjeva da se na ciljanu osobu postave markeri, a ako markera nema ili nisu dovoljno vidljivi zbog kvalitete kamere ili zapreke, sustav nije funkcionalan. Drugi pristup temelji se na oduzimanja pozadine i biblioteci *OpenCV* koja posjeduje dva algoritma koji koriste taj pristup. Uvjeti pod kojima ovaj sustav radi su fiksna pozadina i gibanje objekta koji je potrebno pratiti. Sustav koji je izrađen izdvaja najveći objekt koji se je pojavi na sceni, što konkretno ne mora biti čovjek. Za izdvajanje čovjeka potrebna je dodatna obrada, međutim, iz dosadašnjih rezultata može se zaključiti da je praćenje ključnih točaka ljudskog tijela kao i mjerenje istih dosta zahtjevan zadatak ako se koristi pristup oduzimanja pozadine.

Iz navedenih činjenica moguće je zaključiti da sustavi za praćenje ljudskog tijela koji koriste markere mogu dati zadovoljavajuće rezultate u mjerenju i praćenju. To znači da su u određenim sintetičkim uvjetima, gdje je potrebno postaviti marker za obavljanje određene aktivnosti i gdje vrijeme nije ključan faktor, primjenjivi. Na primjer u robotici ili medicini. Nasuprot tome,

sustavi bez markera mogu poslužiti za globalnu detekciju ili neprecizno i nestabilno praćenje i mjerenje pozicije ljudskog tijela. To znači da je praćenje ljudskog tijela bez markera, dubinskih senzora i s jednom kamerom težak zadatak računalnog vida na čijem se rješenju i dalje treba raditi.

7. LITERATURA

[1] Leonid S. Human pose estimation. Pittsburgh: Disney Research; 2016.

[2] Sood M., Sharma R., Dipakkumar C. Motion Human Detection & Tracking Based On Background Subtraction. International Journal of Engineering Inventions:2013

[3] KaewTraKulPong P., Bowden R.. An Improved Adaptive Background Mixture Model for Realtime Tracking with Shadow Detection. 2nd European Workshop on Advanced Video Based Surveillance Systems; 2001.

[4] Zivkovic Z. Improved adaptive Gaussian mixture model for background subtraction. UK: International Conference Pattern Recognition; 2004.

[5] Kaehler A., Bradski G. Learning OpenCV 3. Sebastopol: O'Reilly Media, Inc.; 2016.

[6] Open Source Coputer Vision. OpenCV 3.2.0. [Online] 23.12.2016. http://docs.opencv.org/3.2.0/

[7] Šribar J., Motik B.. Demistificirani C++: 4. dopunjeno izdanje usklađeno sa standardom C++11/C++14. Zagreb: Element; 2014.

[8] McConnell C.S.. Code Complete: 2. izdanje. Washington: Microsoft Press; 2004.

[9] Bell P., Beer B.. Introducing GitHub. Sebastopol: O'Reilly Media, Inc.; 2015.

[10] Ante Javor. GitHub.[Online] 24.07.2017. https://github.com/antejavor

8. OZNAKE I KRATICE

OpenCV - Open Source Computer Vision library

9. SAŽETAK

Naslov: Vizijski sustav za praćenje poze ljudskog tijela

Ovaj rad obuhvaća temu praćenja pozicije ljudskog tijela. U izradi je korištena popularna biblioteka otvorenog koda *OpenCV*. Biblioteka sadrži mnoštvo funkcija i korisnih klasa koje pomažu u rješavanju problema računalnog vida. U ovom radu opisane su ključne funkcije za razvoj sustava praćenja pozicija ljudskog tijela koje se nalaze u biblioteci *OpenCV*. Pristup oduzimanja pozadine i pristup praćenja *aruco* markera korišteni su za razvoj sustava za praćenje pozicije ljudskog tijela. Sustavi su razvijani pomoću programskog okruženja Visual Studio i programskog jezika C++. Cilj rada je pratiti ključne točke ljudskog tijela i mjeriti iste na što je sustav s markerima dao zadovoljavajuće rezultate.

Ključne riječi: *OpenCV*, praćenje pozicije ljudskog tijela, aruco markeri , oduzimanje pozadine.

10. ABSTRACT

Title: Vision System for Human Body Pose Tracking

The subject of this work is human body tracking. For creating this system is used open source computer vision library, often called *OpenCV*. The library contains many functions and useful classes that are very helpful in solving problems of computer vision. In this work only the most important functions are described that are used in the process of creating vision systems for human body tracking. Methods background subtraction and aruco marker tracking are used to solve the problem of human body tracking. Visual studio an C++ programming language are used to build the system. The goal of this work is to track human body key points and measure them. System that is using the aruco markers has given good results.

Keywords: *OpenCV*, human body pose tracking, aruco markers, background subtraction.

11. PRILOZI

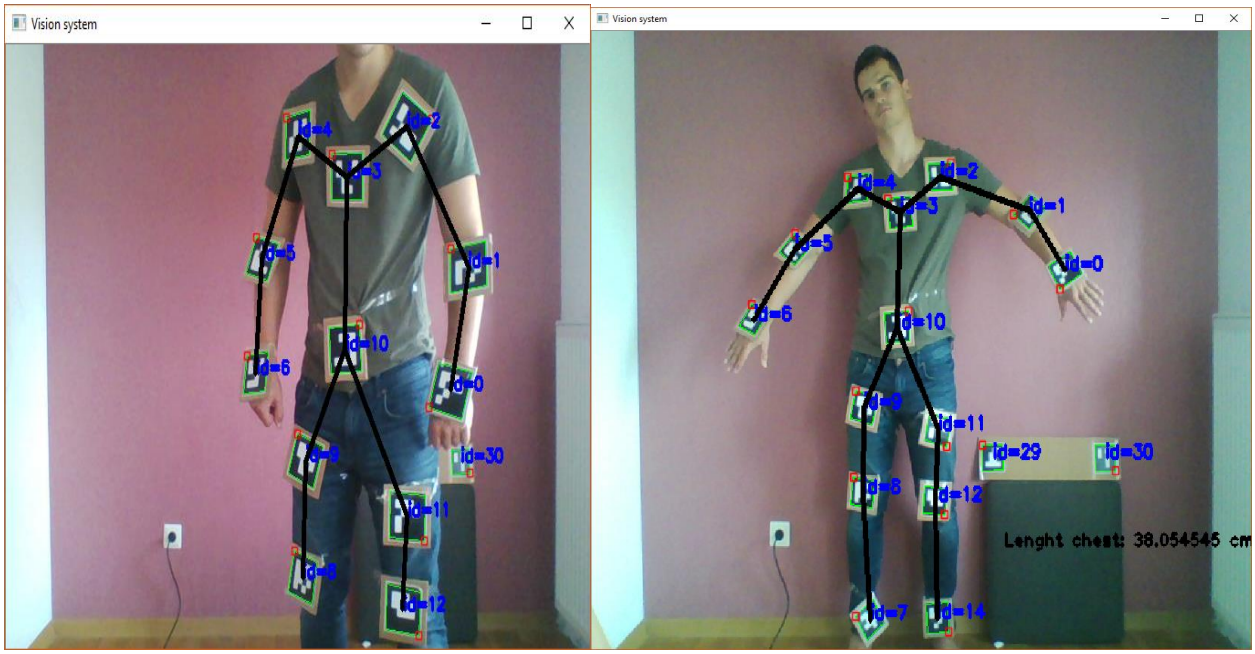
11.1. Programski kod za oduzimanje pozadine

```
/*
*	@file	Main.cpp
*	@author	Ante Javor
*	@date	11.08.2017
*	@brief	Background substraction using MOG2, KNN and drawing the biggest conture
*		and FPS.
*/

#include <opencv2\imgproc.hpp>
#include <opencv2\imgcodecs.hpp>
#include <opencv2\highgui.hpp>
#include <opencv2\video.hpp>

#include <vector>
#include <iostream>
#include <string>
#include <iomanip>

int main()
{
	static int frameCounter;

	cv::Mat frame;

	cv::Mat fgMOG2;
	cv::Mat fgKNN;

	cv::Ptr<cv::BackgroundSubtractor> MOG2;
	cv::Ptr< cv::BackgroundSubtractor> KNN;

/* Creating background substractors with learning rate 2000 frames and showing shadows.
*/
	MOG2 = cv::createBackgroundSubtractorMOG2(2000, 16,true);
	KNN = cv::createBackgroundSubtractorKNN(2000,400, true);

	cv::VideoCapture cap(0);

	while (true) {

		time_t startTime = cv::getTickCount();

		if (!(cap.read(frame)))
			break;

		frameCounter++;
		std::cout << frameCounter << std::endl;

		cv::Mat thresMOG2, thresKNN;
```

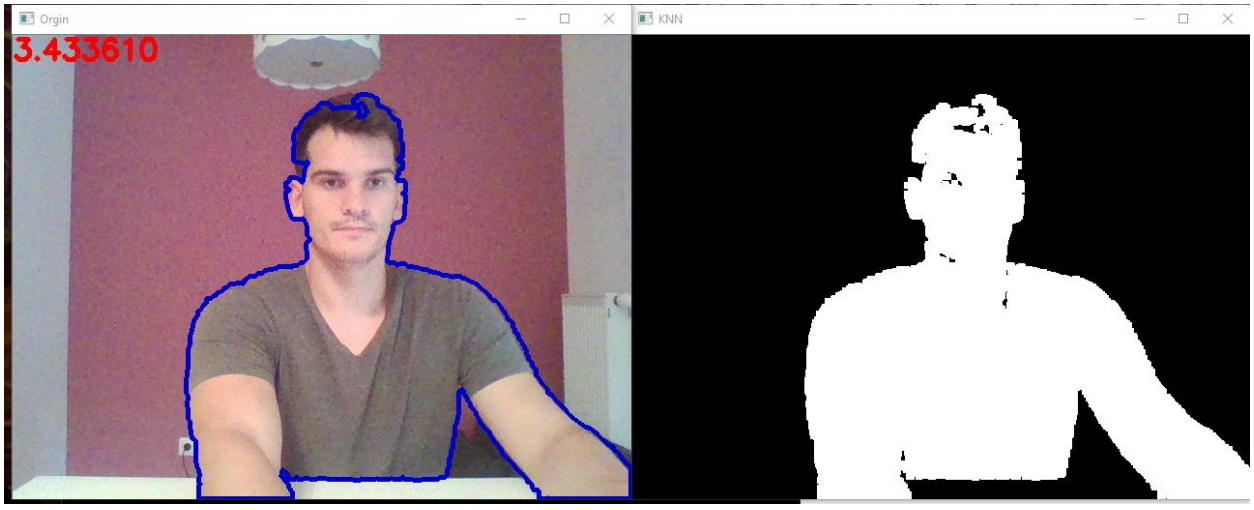
```
		/* Starting background substractors. */
		MOG2->apply(frame, fgMOG2);
		KNN->apply(frame, fgKNN);

		/* Removing shadows from a image. */
		cv::threshold(fgMOG2, thresMOG2, 130, 255, cv::THRESH_BINARY);
		cv::threshold(fgKNN, thresKNN, 130, 255, cv::THRESH_BINARY);

		cv::Mat elementOpen =
cv::getStructuringElement(cv::MorphShapes::MORPH_RECT, cv::Size(3,3));
		cv::Mat elementDilate =
cv::getStructuringElement(cv::MorphShapes::MORPH_RECT, cv::Size(5,5));

	/* Removing image noise with morphology operation open and dilate. */
		cv::Mat openMOG2, openKNN;
		cv::morphologyEx(thresMOG2, openMOG2, cv::MORPH_OPEN,
elementOpen,cv::Point(-1,-1),1);
		cv::morphologyEx(thresKNN, openKNN, cv::MORPH_OPEN, elementOpen,
cv::Point(-1, -1), 1);
		cv::Mat dilateMOG2, dilateKNN;
		cv::dilate(openMOG2, dilateMOG2,elementDilate);
		cv::dilate(openKNN, dilateKNN,elementDilate);


		std::vector<std::vector<cv::Point>> conturs;
		cv::findContours(dilateKNN, conturs, cv::RETR_EXTERNAL,
cv::CHAIN_APPROX_NONE);

		/* Extracting contur ID with biggest area. */
		int largestArea = 0;
		int conturIndex = 0;
		for (int i = 0; i < conturs.size(); i++)
		{
			double area = cv::contourArea(conturs[i]);
			if (area > largestArea)
			{
				largestArea = area;
				conturIndex = i;
			}
		}

		cv::drawContours(frame, conturs, conturIndex, cv::Scalar{ 200,0,0 }, 3, 8);

		if (cv::waitKey(1) == 27)
			break;

		/* Calculating frames per second. */
		time_t endTime = cv::getTickCount();
		double framesPerSecond = (cv::getTickFrequency() / (endTime - startTime));
		std::string framesPerSecondString = std::to_string(framesPerSecond);

		/* Put frames per second on image. */
		cv::putText(frame,framesPerSecondString, cv::Point(0, 25),
cv::FONT_HERSHEY_SIMPLEX, 1, cv::Scalar{ 0,0,255 }, 3);

		cv::imshow("Orgin",frame);
		cv::imshow("MOG2", dilateMOG2);
		cv::imshow("KNN", dilateKNN);
```

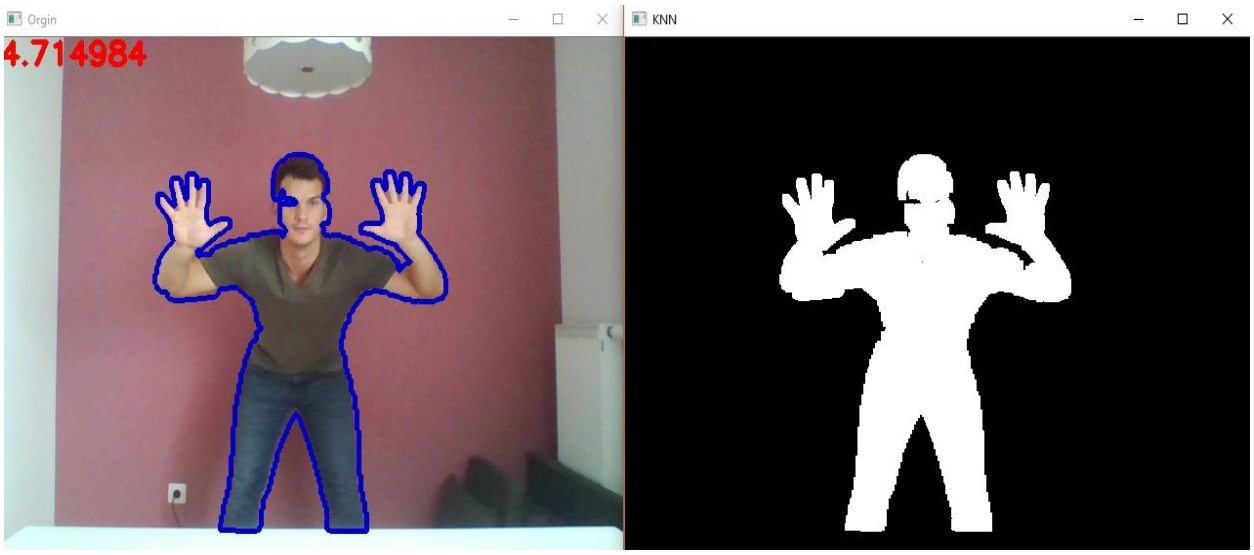
```
        }
        return 0;
}
```

11.2. Rezultati praćenja čovjeka *aruco* markerima



11.3. Rezultati praćenja čovjeka oduzimanjem pozadine




Orgin
4.714984
KNN

I Z J A V A O A U T O R S T V U Z A V R Š N O G R A D A

Pod punom odgovornošću izjavljujem da sam ovaj rad izradio/la samostalno, poštujući načela akademske čestitosti, pravila struke te pravila i norme standardnog hrvatskog jezika. Rad je moje autorsko djelo i svi su preuzeti citati i parafraze u njemu primjereno označeni.

Mjesto i datum	Ime i prezime studenta/ice	Potpis studenta/ice
U Bjelovaru, 11.09. 2017	Ante Javor	

Prema Odluci Visoke tehničke škole u Bjelovaru, a u skladu sa Zakonom o znanstvenoj djelatnosti i visokom obrazovanju, elektroničke inačice završnih radova studenata Visoke tehničke škole u Bjelovaru bit će pohranjene i javno dostupne u internetskoj bazi Nacionalne i

sveučilišne knjižnice u Zagrebu. Ukoliko ste suglasni da tekst Vašeg završnog rada u cijelosti bude javno objavljen, molimo Vas da to potvrdite potpisom.

Suglasnost za objavljivanje elektroničke inačice završnog rada u javno dostupnom nacionalnom repozitoriju

Ante Javor

ime i prezime studenta/ice

Dajem suglasnost da se radi promicanja otvorenog i slobodnog pristupa znanju i informacijama cjeloviti tekst mojeg završnog rada pohrani u repozitorij Nacionalne i sveučilišne knjižnice u Zagrebu i time učini javno dostupnim.

Svojim potpisom potvrđujem istovjetnost tiskane i elektroničke inačice završnog rada.

U Bjelovaru, 11.09.2017

potpis studenta/ice