Upravljanje IP PTZ kamerom temeljeno na semantičkim mrežnim usluga za praćenje promjena u sceni

Lončar, Sven

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Sven Lončar

Zagreb, 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentor:
doc. dr. sc. Tomislav Stipančić

Student:
Sven Lončar

Zagreb, 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za diplomske radove studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala te mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum:	Prilog:
Klasa: 602-14/22-6/1	
Ur. broj: 15-1703-22-	

DIPLOMSKI ZADATAK

Student: **SVEN LONČAR** Mat. br.: 0035207261

Naslov rada na hrvatskom jeziku: **Upravljanje IP PTZ kamerom temeljeno na semantičkim mrežnim usluga za praćenje promjena u sceni**

Naslov rada na engleskom jeziku: **IP PTZ camera control based on semantic network services for monitoring changes in the scene**

Opis zadatka:

Semantičke mrežne tehnologije se temelje na mrežnom komunikacijskom protokolu SOAP (eng. Simple Object Access Protocol) te zapisnicima definiranim u programskom jeziku WSDL (eng. Web Services Description Language) koji sadržavaju definicije i opise funkcionalnosti uređaja na mreži. Uređaji čiji je rad temeljen na semantičkim mrežnim uslugama mogu s manjim ili većim stupnjem autonomnosti dijeliti dostupne informacije te izvršavati željene aktivnosti kako u virtualnom, tako i u realnom svijetu. Protokoli semantičkih mrežnih tehnologija pritom moraju omogućiti siguran i pravovremen prijenos informacija.

U zadatku je potrebno:

- razviti programsku podršku za mrežnu kameru (oznaka: PTZ – Pan, Tilt, Zoom) koja osigurava praćenje objekta koji se kreće trenutno mirnom okolinom,
- ostvariti komunikaciju između računala tražitelja usluge i mrežne kamere (oznaka: IP) s tri stupnja slobode gibanja (oznaka: PTZ – Pan, Tilt, Zoom) temeljenu na semantičkim mrežnim uslugama, poštujući pritom zahtjeve u pogledu sigurnog prijenosa podataka i privatnosti.

Razvijenu programsku podršku potrebno je eksperimentalno verificirati na opremi dostupnoj u sklopu Laboratorija za projektiranje izradbenih i montažnih sustava koja uključuje IP PTZ kameru i upravljačko računalo sa svom potrebnom mrežnom opremom.

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

Zadatak zadan: 5. svibnja 2022.

Rok predaje rada: 7. srpnja 2022.

Predviđeni datum obrane: 18. srpnja do 22. srpnja 2022.

Zadatak zadao: doc. dr. sc. Tomislav Stipančić

Predsjednica Povjerenstva: prof. dr. sc. Biserka Runje

IZJAVA

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

Zahvaljujem mentoru *doc. dr.sc Tomislavu Stipančiću* i *Leonu Korenu, mag.ing.mech.*, na savjetima, uloženom vremenu, primjedbama i korekcijama koje su pridonijele boljitku i kvaliteti ovog rada.

Zahvalu upućujem i svojoj obitelji i prijateljima za nesebičnu podršku koju su mi pružili tijekom čitavog školovanja.

U Zagrebu, 20. veljače 2020.

_____

Sven Lončar

SAŽETAK

Ovaj rad će se baviti povezivanjem računala i kamere putem internetske mreže i specifičnih protokola koji služe za izmjenu informacija. Kamera šalje sliku računalu, računalo obrađuje informacije, donosi određene zaključke putem dobivenih informacija te šalje skup podataka kameri, odnosno naredbe koje će kameri govoriti kada i gdje se mora gibati te kada mora stati. Računalo će primati sliku od kamere, obrađivati sliku pomoću napisanog programa koji se bavi računalnim vidom te pomoću napisanog koda otkriti gdje se nalazi promjena na slici. Na temelju pozicije kamere, referentnih točaka u prostoru te pozicije čovjeka na slici, program radi analizu podataka te kao završnu informaciju odlučuje smjer u kojem se kamera mora gibati kako bi čovjeka stavila u svoj centar slike.

SUMMARY

This paper is going to be about connecting computers and cameras through internet network and specific protocols which serve for an exchange of information. Camera is sending frames to the computer, computer is processing the information, makes certain conclusions through received informations and is sending a package of data and functions to the camera which will tell the camera where and when it has to move and when to stop. Computer will receive frames from the camera and it will analyze them through the written code. The code is written with the help of computer vision techniques and it will detect changes or motion on the images. Based on the position of the camera, reference points in the space and the position of the object on the images, program is making data analysis and as a final information it decides in which direction camera should be moving to put the object in to the frame center.

SADRŽAJ

POPIS SLIKA

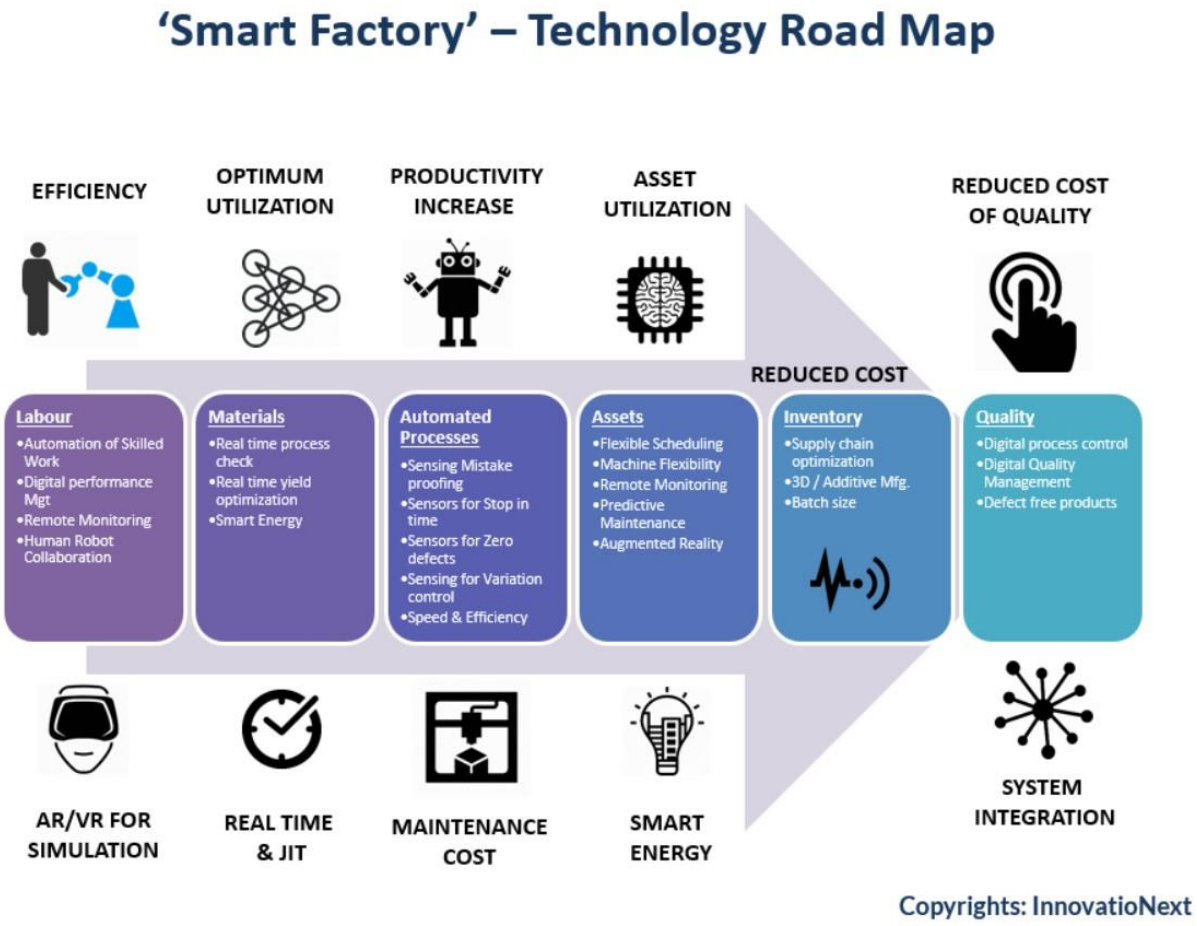
1. UVOD

Razvoj tehnologije i digitalizacija procesa dovodi do novih potreba u društvu sa ciljem napretka i automatizacije. Sve veći i brži razvoj industrije postavlja nove zahtjeve i vizije prema kojima bi se društvo i razvoj trebali usmjeravati kako bi se ostvarili postavljeni ciljevi. Industrijski pogoni teže sve većoj autonomiji i digitalizaciji procesa u kojima će računala obavljati određene poslove te obradu velikih količina podataka. Repetitivni poslovi se sve više zamjenjuju računalima i robotima zbog svoje točnosti, pouzdanosti te brzine. To omogućuje da se ljudski resursi usmjeravaju u druga područja gdje je primjena računala i autonomnih sustava onemogućena. Današnja industrijska revolucija, odnosno industrija 4.0, ide u smjeru povezivanja pojedinačnih dijelova sustava u jednu cjelinu gdje su ti dijelovi u međusobnoj komunikaciji te razmjenjuju informacije sa ciljem prilagođavanja sustava trenutnim uvjetima i zahtjevima okoline. Pojam industrija 4.0 se koristi od 2011. godine za opisivanje rasprostranjene integracije informacijskih i komunikacijskih tehnologija u industrijskoj proizvodnji. Susrećemo se sa pojmovima poput pametnog uređaja ili pametne tvornice. Takvi sustavi i proizvodi prolaze kroz transformacijske procese gdje se njihova unaprijeđenja odnose na digitalnu transformaciju unutar proizvodnih procesa. Transformacija se temelji, kao što je već rečeno, na inteligentnom umrežavanju strojeva i drugih uređaja pomoću naprednih komunikacijsko – informacijskih tehnologija kako bi se omogućilo autonomno komuniciranje među uređajima, prikupljanje velike količine podataka te njihova obrada i analiza, autonomno donošenje odluka, praćenje raznih procesa u sustavu u realnom vremenu, stvaranje dodane vrijednosti te vertikalna i horizontalna integracija. [1]

Tehnologije o kojima ovisi razvoj ovakvih naprednih sustava su brojni. Ključan pojam za industriju 4.0 je IoT (*"internet of things").* On je, može se reći, najvažniji element takvih sustava. Njegova karakteristika je povezivanje uređaja sa ciljem komunikacije sa bazom, središnjim računalom preko kojega je moguća interakcija čovjeka sa sustavom. Podaci sa strane uređaja preko određenog komunikacijskog protokola dolaze na računalo koje tada izvršava analizu podataka, donosi odluke temeljene na dobivenim podacima te je prikupljene podatke, izvršene odluke i predviđanja moguće predočiti čovjeku na razumljiv način. Druga novina koje se također usko veže uz takve sustave je "*cloud computing"* (oblačno računarstvo, rad u oblaku). Ono omogućava pristupanje računalnim resursima, na zahtjev, odnosno korištenje raznih računalnih resursa kao što su npr. server, aplikacije, pohrana podataka, korištenje razvojnih alata ili korištenje raznih mrežnih mogućnosti. Taj "*cloud"* je udaljen podatkovni centar koji omogućuje korištenje njegovih usluga putem mreže. Još neke tehnologije koje su karakteristične za takve sustave su napredna robotika, gdje su roboti opremljeni raznim senzorima kako bi mogli slati informacije o robotu, proširena i virtualna stvarnost, digitalni blizanci, koji mogu rekreirati industrijski pogon ili

proces u virtualnom okruženju gdje je moguće vizualno pratiti sve procese u pogonu u realnom vremenu, simulacije te brojne druge. [2]

Cilj ovih tehnologija je poboljšanje već postojećih proizvodnih procesa te kreiranje novih i naprednih. Unos istih količina resursa kao što su vrijeme, radna snaga te novac uz korištenje ovih tehnologija rezultira povećanjem produktivnosti, skraćuje vrijeme potrebno da se proizvod plasira na tržište, povećava kvalitetu proizvoda i usluga, smanjuje cijenu proizvodnje, optimizira potrošnju energije te služi za dijagnostiku i nalaženje određenih procesa koji bi se dalje mogli unaprijediti kako bi se sami procesi unutar sustava još poboljšali.



Slika 1 - Prednosti pametnih tvornica [3]

Jedna od novijih tehnologija, a koja se danas sve više razvija i poprima široke primjene u industriji, je računalni vid. To područje računarstva je predmet ovog rada zajedno sa mrežnom komunikacijom između završnog uređaja – kamere, te računala koje vodi sve procese.

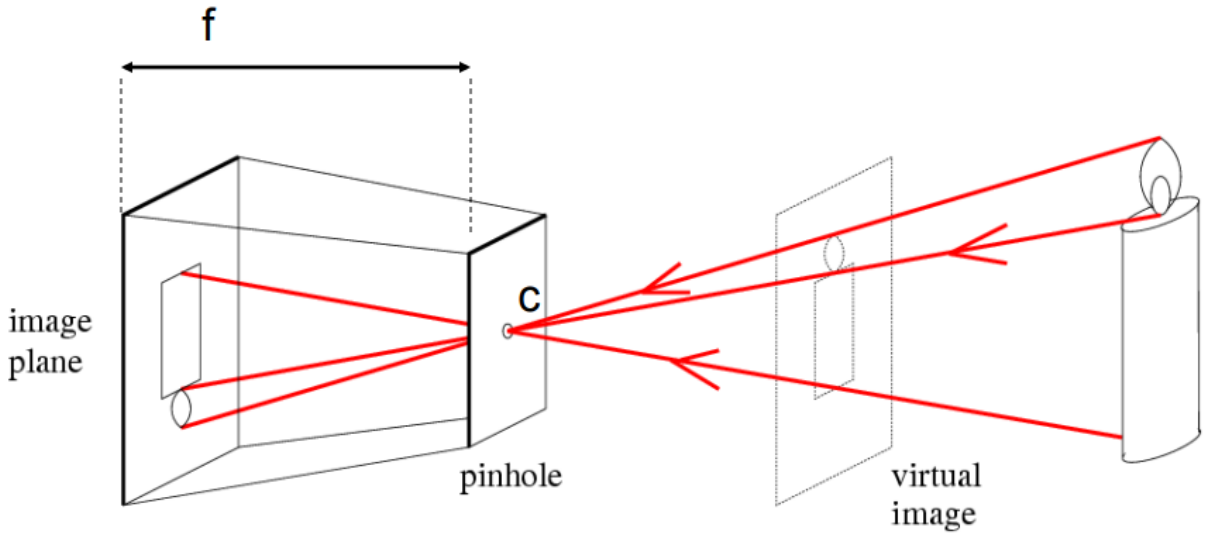
2. RAČUNALNI VID

2.1. Uvod

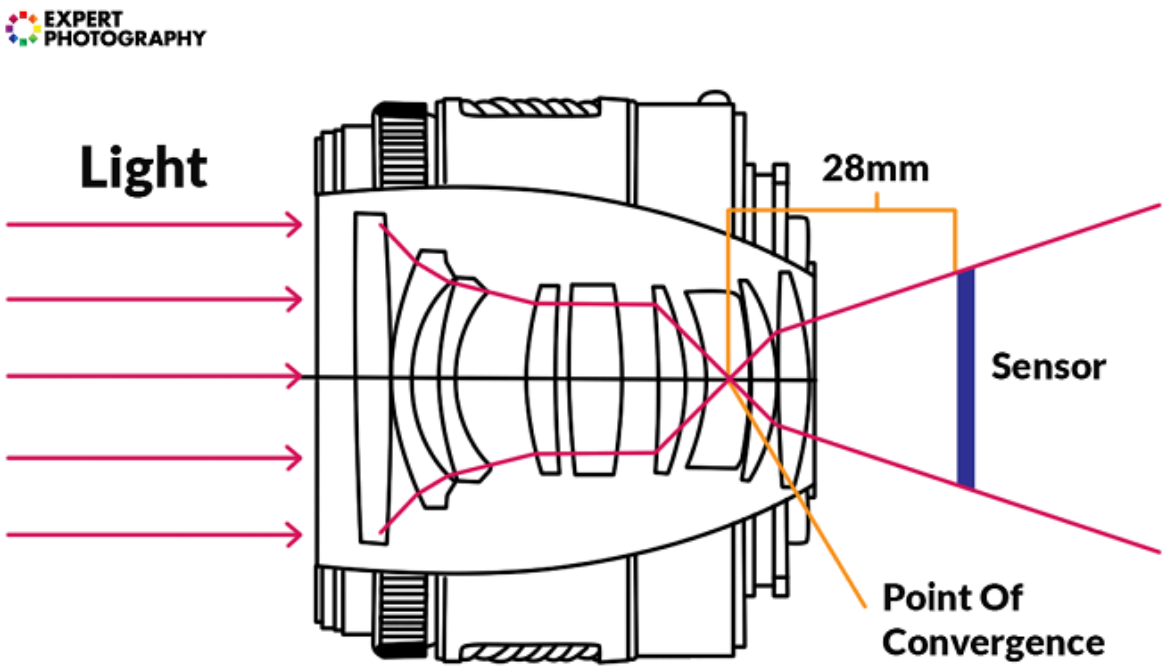
Jedno od najbitnijih ljudskih osjetila za percipiranje prostora oko nas je vid. Gledajući prostor oko sebe čovjek poprima informacije o objektima, udaljenostima, dubini prostora, osvijetljenosti itd. Fotoni preko oka, mrežnice, fotoreceptora i brojnih živaca dolaze u mozak. Preklapanjem dviju slika, po jedna iz svakog oka, stvara se konačna slika pomoću koje dobivamo i percepiju dubine uz sve ostale informacije. Čovjek se oduvijek koristio vidom kao glavnim osjetilom za preživljavanje, odnosno za lociranje hrane, vode ili prepoznavanje opasnosti. Kako se društvo sve više razvijalo, čovjek je počeo čitati, crtati, vidom provjeravati predmet rada. Daljnjim razvitkom društva i industrije čovjek se našao u pogonima gdje radi repetitivne zadatke poput provjere kvalitete proizvoda na proizvodnoj traci ili sortiranje proizvoda u pogonu. Tragajući za boljim riješenjima, podizanjem efikasnosti proizvodnje i ubrzanju cjelokupnog procesa proizvodnje, u 20. stoljeću se počinje razvijati područje računarstva zvano računalni vid.

2.2. Stvaranje slike

Analogno stvaranju slike u mozgu, gdje zrake padaju na očnu leću, te zatim iza toga na mrežnicu, u kojoj se dobiva obrnuta slika, dobivanje slike na kameri je sličan proces pojednostavljeno prikazan na *Slika 2*.



Slika 2 - Pojednostavljeni prikaz dobivanja slike na kameri [4]

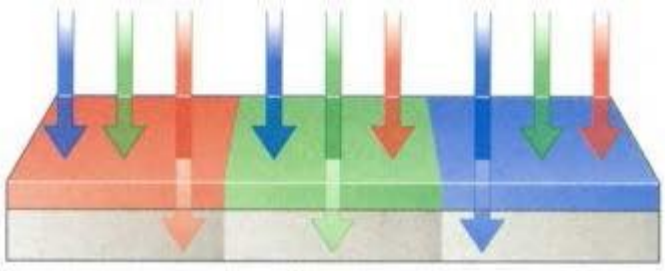


Slika 3 - Prolaz zraka svjetlosti kroz objektiv kamere [5]

Predmet koji se nalaze ispred kamere, u osvijetljenom prostoru, odbija svjetlosne zrake, odnosno fotone u prostoru oko sebe. Zrake su shematski prikazane na Slika 2. i Slika 3. Zrake padaju na leću kamere te se daljnjim putem spajaju u jednu točku koja se zove točka konvergencije. Na Slika 2. je ta točka označena slovom *C*, a na Slika 3. je označena sa tekstualnom oznakom "*Point of convergence*". Točka konvergencije je točka u prostoru iza koje dolazi do obrtanja slike. Iza točke konvergencije se nalazi film na koji obrnute zrake svjetlosti, odnosno slika, padaju te se na taj način stvara slika.

Današnji fotoaparati ne koriste fotosenzibilne filmove koji kemijskim reakcijama stvaraju sliku, već su se razvili digitalni senzori. Ti senzori u sebi sadrže fotosenzibilne ćelije koje u dodiru sa fotonima daju signale. Ispred senzora se nalaze filteri koji propuštaju samo svoji spektar svijetlosti na senzor. Tako propuštena svijetlost pada na senzor te se generira električni impuls, odnosno signal. Intenzitet boje na slici je produkt količine svijetlosti koja padne na senzor. Takav električni impuls, odnosno intenzitet boje se zapisuje u digitalnom obliku, u obliku broja. Pred senzorom se nalaze tri filtera: crveni, plavi i zeleni, te se kombinacijom intenziteta njihovih svijetlosti dobiva slika u boji. Filter će biti prikazan na Slika 4. Na Slika 5. je prikazano propuštanje određenog spektra svijetlosti kroz određeni filter te na Slika 6. su prikazana 3 dobivenja zapisa boja, po 1 za svaki filter. Njihovim kombiniranjem te interpolacijom dobivamo potpunu sliku u boji. [6]

Slika 4 – Filterska mreža ispred senzora [5]



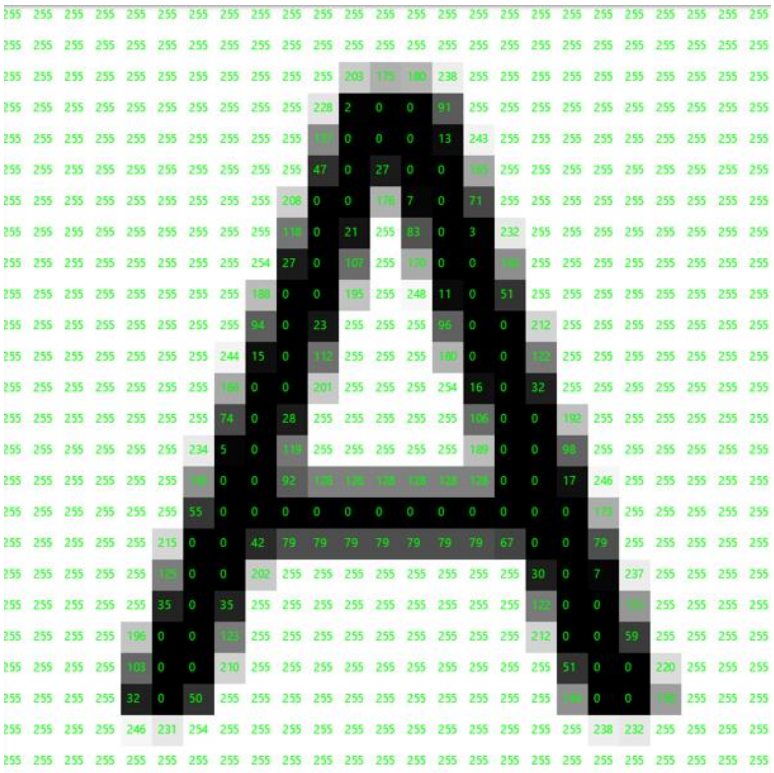
Slika 5 - Propuštanje određenog spektra svijetlosti kroz određeni filter [5]

Slika 6 – Zapis slike korištenjem pojedinačnih filtera [5]

Filterska se mreža najčešće postavlja po Bayerovom modelu. To je metoda kreiranja filter mreže u kojoj se u određenom redoslijedu postavljaju filteri gdje 50% površine čini zeleni filter te po 25% crveni i plavi.

2.3. Slika u digitalnom obliku

Slika koju vidimo na ekranu i elektronskim uređajima mora biti negdje pohranjena sa svim potrebnim informacijama kako bi uređaj mogao reproducirati sliku u svom izvornom obliku. Svaki dio slike posjeduje svoju boju i intenzitet te odgovarajuću informaciju, odnosno skup podataka, pohranjenu na računalu za taj dio slike. Kako bi ekran nekog uređaja uspio prikazati više od jedne informacije, potrebno je razložiti njegovu površinu na što više dijelova kako bi prikazana slika bila sličnija originalu. Najmanji osnovni element kod prikaza slike se naziva piksel. Piksel je mali dio površine slike koji ima sposobnost prikaza određene boje. Što je na slici više piksela, to je moguće prikazati više boja, više informacija te je slika samim time kvalitetnija. Svaki piksel na slici prezentira određeni dio prostora originalne slike pa je lako zaključiti što je piksel manji i što ih je više, to će slika biti sličnija izvornoj.



Slika 7 - Pikseli na slici

Mjera koja daje informaciju o količini piksela na slici se naziva rezolucija. Ona može biti definirana na mnogo načina, a za ovaj rad je bitna pikselska rezolucija. Pikselska rezolucija govori kolko je piksela na slici, odnosno koliko se piksela nalazi u jednom retku po širini slike te koliko se piksela nalazi u jednom stupcu po visini slike. Većina ljudi se susrela sa ovim pojmom kod svojih računalnih monitora. Veličina rezolucije je uglavnom

standardizirana što se tiče monitora te se kreće u vrijednostima 640x480, 800x600, 1024x768 i tako dalje, a zajednički im je omjer koji iznosi 4:3.

Slika u digitalnom obliku može biti zapisana na više načina. Postoji rasterski oblik zapisa slike i vektorski. Vektorski oblik zapisa slike sadrži formule koje opisuju ono što se prikazuje na slici. Rasterski oblik zapisa je dvodimenzionalnog oblika, odnosno to je matrica u kojoj su zapisane informacije o slici. Vektorski oblik zapisa prilikom neke promjene na slici, npr. kada zumiramo vektorsku sliku, ponovno se kalkuliraju oblici koji će se prikazati. Najpoznatiji takav tip zapisa je PDF dokumet. Kada se zumira jedna stranica PDF dokumenta možemo vidjeti kako će se slova svaki puta izoštriti za razliku od rasterskog zapisa slike. Izoštravanje u PDF dokumentu je prikazano na Slika 8. i Slika 9.

Slika 8 - Normalan prikaz dijela PDF dokumenta

Slika 9 - Povećani prikaz dijela PDF dokumenta

Na slici 8. vidimo slova normalnih rubova. Nema sporih prijelaza sa crne na bijelu boju kao što će biti kod prikaza rasterskog oblika na slici 10. i slici 11. Na slici 9. vidimo povećani prikaz istog dijela vektorskog zapisa gdje su rubovi također oštri što je posljedica ponovnih kalkulacija vektorskih zapisa prilikom svake manipulacije vektorske slike.

Kod rasterskih slika je slučaj drugačiji. Kako je rasterski oblik zapisa konačna 2D matrica sa svojstvenim vrijednostima koje su uvijek iste, tako je i konačan zapis slike koji nema mogućnost promjene prilikom manipulacije slike. Prikaz je dan na slici 10. I slici 11.

Slika 10 - Normalan prikaz rasterske slike

Slika 11 - Povećani prikaz rasterske slike

Na slici 11. je uočljiva razlika sa slikom 9. Na slici 11. vidimo kako rubovi nisu tako oštri već se vidi postupni prijelaz sa crne na bijelu boju. Uočljive su zone sive boje na rubu slova. Daljnjim povećanjem slike 11. uočeni dijelovi bi bili još izraženiji.

Iako se čini da je vektorski zapis slike bolji od rasterskog, rasterski je jednostavniji za interpretaciju podataka. Rasterski oblik slike se najviše koristi u računalnom vidu zbog lakoće iščitavanja podataka i analize istih. Izgled jednostavne rasterske slike u crno-bijelom načinu prikaza slike te pripadajuća matrica su dani na slici 12.

1	3	0	0	1	12	8	0
1	4	3	3	0	2	0	2
1	7	4	1	5	4	2	2
0	3	1	2	2	2	2	3
0	5	1	9	3	3	3	4
5	0	8	0	2	4	3	2
8	4	3	2	2	7	2	3
2	10	1	5	2	1	3	7

Slika 12 - Zapis slike u matričnom obliku i njen prikaz u bojama

Zbog jednostavnosti razumijevanja slika 12. je izvedena tako da područja sa većim brojevima imaju tamnije nijanse sive boje dok područja sa nižim vrijednostima imaju svijetlije nijanse sive. Kako i na koji način se zapravo interpretiraju vrijednosti će biti objašnjeno u narednom poglavlju o sustavima boja.

Postoje razni oblici zapisa rasterskih slikovnih datoteka u računalu. Svaki oblik sadrži neke prednosti i mane pa se prema tome svaki koristi za određene primjene, zavisno o zahtjevima procesa za koji će se koristiti slike. Neki od rasterskih vrsta formata slika su JPEG,

PNG, RAW, TIFF, GIF, HEIF i brojni drugi. Postoje i tipovi kompresija slika sa ciljem štednje memorije, odnosno smanjenja količine podataka potrebnih za pohranjivanje slike. To su *lossless* i *lossy* kompresija slike. "*Lossless"* tip kompresije je vrsta smanjenja količine podataka gdje ne dolazi do gubitka informacija sa slike te se koristi uglavnom za arhiviranje slika. "*Lossy"* tip kompresije znantno smanjuje veličinu memorije koju zauzima slika ali uz to i smanjuje informacije sa slike. Dovodi do smanjenja informacija ali u tolikoj mjeri da se slika i dalje može koristiti u određene svrhe.

JPEG (*joint photographic experts group*) je rasterski tip slike, značajan je po tipu *lossy* kompresije čime mu je mala veličina ali i kvaliteta slike je oslabljena. To je ujedno i najrašireniji tip slike jer je pogodan za dijeljenje i prijenos. Najčešće se koristi za prikaz u internet preglednicima gdje je postao univerzalan zapis slike. PNG (*portable network graphics)* je tip rasterske slike koji je značajan po *lossless* tipu kompresije. Bolje zadržava detalje i kontraste boja od JPEG tipa podatka čime ujedno zauzima i više memorije za pohranu. GIF tip slike je najpoznatiji z aprikaz animiranih slika. Pogodan je za to jer smanjuje raspon boja i koristi 8-bitova memorije za prikaz piksela čime mu se drastično smanjuje veličina za pohranu slike. TIFF (*tagged image file format*) je tip slike koji se ne kompirmira te je zato pogodan za pohranu i uređivanje slika. HEIF (*high efficiency image format*) je tip slike razvijen sa ciljem da bude konkurentan JPEG tipu slike. Za drugu metodu kompresije slika bude iste veličine kao JPEG ali posjeduje puno više informacija koje se ne izgube tokom kompresije. [8]

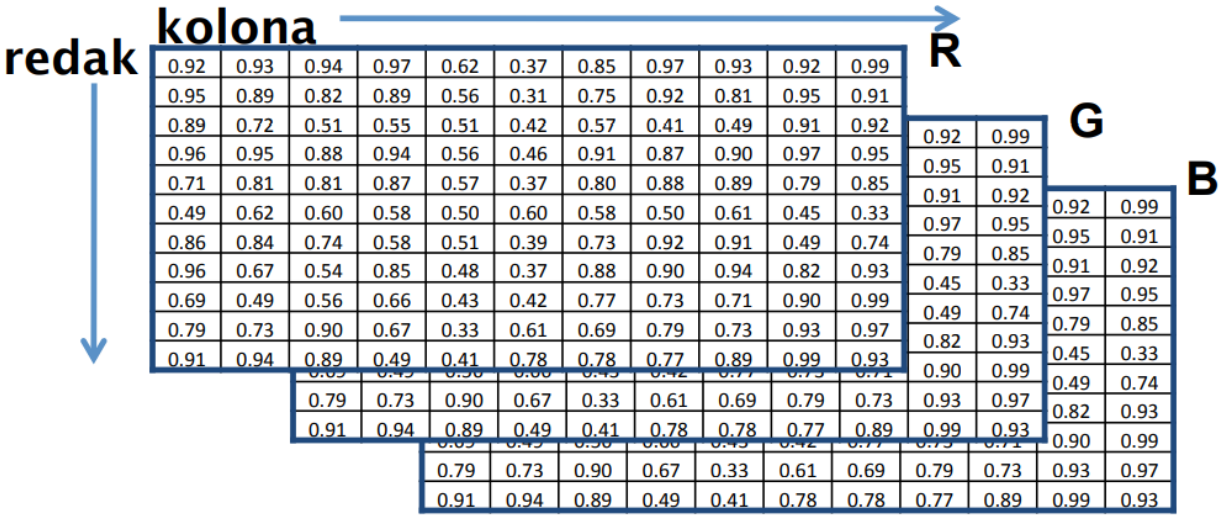
Slika 13 - usporedba HEIC i JPG slike [9]

2.4. Sustavi boja

Danas postoje razvijeni mnogobrojni sustavi boja. Svaki nalazi svoju primjenu u nekom području. Okom vidljivi spektar boja zavisi od valne duljine svijetlosnih valova koji putuju prostorom oko nas. U fizikalnom svijetu svaka boja ima svoju valnu duljinu, svoj specifični iznos valne duljine, brojčani iznos. Specifičan iznos intenziteta određene veličine ili veličina u sustavu boja definira neku boju. Svaka boja ima svoj zapis u određenom obliku u pojedinom sustavu boja koji karakterizira samo tu boju. Svaki sustav boja opisuje boje na sebi svojstven način. Neki od poznatijih sustava boja koji se danas primjenjuju su RGB, HSV, YCbCr, CMYK, monokromatski (za ovaj rad najvažniji *grayscale)* i mnogi drugi. U tiskarskoj industriji printeri najčešće koriste navedene 4 boje (CMYK) koje u određenim kombinacijama daju sve ostale boje. YCbCr je izveden tip sustava boja od RGB-a koji služi za kompresiju slike ili videa zbog lakšeg i bržeg prijenosa podataka. RGB, HSV i *grayscale* su korišteni u obradi slike, odnosno u računalnom vidu te će o njima ovdje biti riječ.

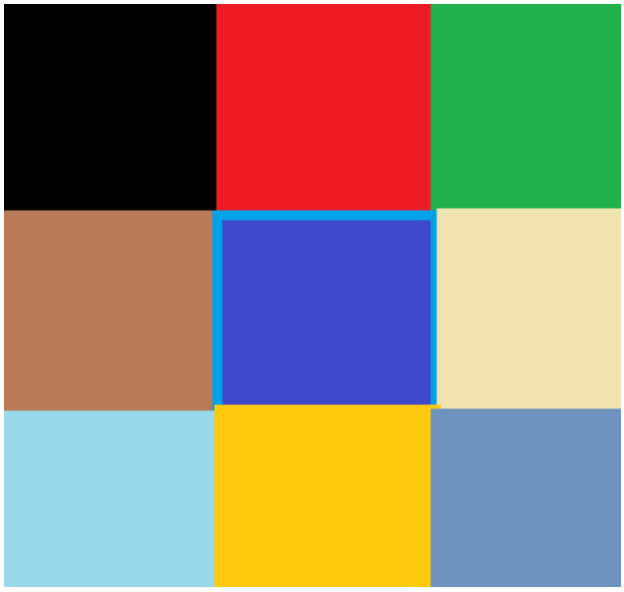
2.4.1. RGB

RGB (*red, blue, green*) sustav se temelji na 3 boje pomoću kojih se prezentira slika u boji, a to su plava, crvena i zelena. RGB sustav, može se zaključiti iz prethodnih poglavlja, koristi se npr. u senzorima digitalnih kamera za dobivanje slike. Ovaj sustav u digitalnom obliku za prikaz slike koristi 3 matrice u kojoj svaka od njih sadrži intenzitete piksela određene boje. Preklapanjem te 3 matrice dolazi do komputacije boja te se prikazuje određena boja na pikselu.



Slika 14 - Primjer digitalnog zapisa RGB slike [4]

RGB sustav boja je standardiziran i najčešće korišten u svim digitalnim oblicima prikaza slika. Internet preglednici koriste RGB sustave boja kao standard. Intenziteti boja se prikazuju brojevima od 0 do 255. Do toga je doveo sustav bitova, odnosno pohrana memorije u komponentama računala. Računalo razumije samo signale odnosno nule ili jedinice. Njih pohranjuje u nizove od 8 bitova. Kako je moguće pohraniti 2^8 različitih kombinacija nula ili jedinica, odnosno 256 raličitih zapisa, to dovodi do mogućnosti pohrane 256 različitih boja. Kako programski jezici počinju brojati od 0 to znači da se boje zapisuju od 0 do 255. S obzirom da se u RGB-u boje zapisuju pomoću 3 informacije, po jedan broj od 0 do 255 za crvenu, plavu i zelenu matricu to znači da računalo može prikazati 256*256*256 različitih boja, odnosno 16 777 216 boja. Za bolje razumijevanje će biti objašnjeno kako se dobiju neke od osnovnih boja. Za RGB vrijednosti 255,0,0 boja piksela će biti crvena. Analogno tome, ako piksel ima vrijednosti 0,255,0 boja će biti zelena te ako piksel ima vrijednost 0,0,255 boja će biti plava. Za vrijednosti 0,0,0 boja će biti crna, a za 255, 255, 255 boja će biti bijela. S obzirom da će se u ovom radu raditi isključivo sa RGB i *grayscale*-om koji je poprilično sličan RGB-u, biti će prikazan princip rada. Kreirana je jednostavna slika 15. te će biti prikazane njene vrijednosti u slici 16. Slika je kreirana u programskom paketu *Paint* koji dolazi uz *Windows* operativni sustav, a ostale operacije su izvršene u programskom jeziku *Python*-u.



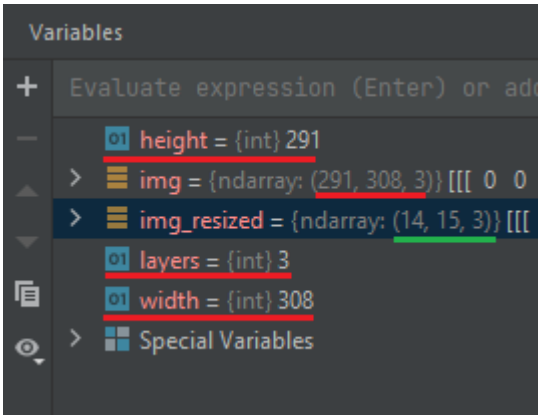
Slika 15 - RGB slika

Veličina originalne slike iznosi 308x291 px. Radi preglednosti podataka slika će biti smanjena 20 puta te će se onda iščitati vrijednosti piksela. S obzirom da je slika opisana sa 3 matrice, vrijednosti piksela će biti prikazane za pojedine redove slike gdje ćemo za svaki piksel imati 3 vrijednosti. Kada sliku smanjimo 20 puta, njena veličina će biti 15x14 px - . 15x14 znači da ćemo imati 15 stupaca i 14 redova u matrici. S obzirom da je slika RGB,

program bi trebao pokazati 3 vrijednosti za svaki piksel. Dobiveni rezultat je prikazan na slici 17.

```
import cv2 as cv

img = cv.imread('C:\\Users\\CAD\\Documents\\Diplomski\\boje.png')

height, width, layers = img.shape   height: 291    layers: 3    wid
img_resized = cv.resize(img, (int(width/20), int(height/20)))   img
cv.imshow('img', img_resized)
cv.waitKey(0)
```

Slika 16 - Primjer programskog koda za dobivanje podataka o slici



Slika 17 - Pregled varijabli iz programskog koda

Na slici 17. su vidljive varijable dobivene programskim kodom sa slike 16. Učitali smo sliku 15. u *Python* te htjeli dobiti veličinu slike. Crvenom crtom podcrtane su veličine koje se odnose na originalnu sliku. Vidimo da je visina slike (*height)* 291 piksel, a širina slike (*width)* 308 piksela. Broj slojeva *(layers)* je 3 što označava broj matrica kojima se opisuje slika. Zelenom crtom su podcrtani podaci za smanjenu sliku iz koje se vidi veličina slike 15x14 te broj slojeva 3.

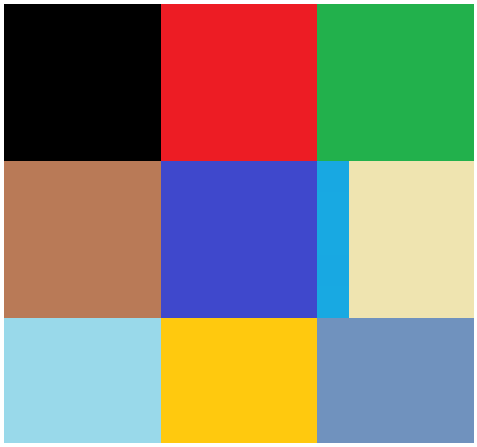
U prvom retku piksela slika sadrži 3 boje: crnu, te crvenu i zelenu u određenim nijansama. To znači da bi u sve 3 matrice prvi redak trebao biti popunjen sa 3 različite vrijednosti brojeva. Dobiveni brojevi prikazani su na slici 18.

	0	1	2
0	0	0	0
1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	0
5	36	28	237
6	36	28	237
7	36	28	237
8	36	28	237
9	36	28	237
10	76	177	34
11	76	177	34
12	76	177	34
13	76	177	34
14	76	177	34

Slika 18 - Vrijednosti prvog retka piksela sa umanjene slike 15.

Sa slike 18. vidimo listu od 0 do 14 koja predstavlja piksele u jednom retku slike. Za svaki piksel imamo 3 vrijednosti u stupcima 0, 1 i 2. Oni predstavljaju intenzitete pojedinih R, G i B matrica. Kako smo i očekivali, vidimo da svaka matrica posjeduje samo 3 različita broja jer se u prvom retku piksela slike nalaze samo 3 različlite boje.

U sredini slike 15. vidimo 4 različite boje te stoga očekujemo 4 različita broja koji će mijenjati položaje u odnosu na pojavljivanje određene boje. Dobiveni rezultati će biti prikazani na slici 20. Zbog velike kompresije slike očekujemo i gubitke na slici, odnosno kako vidimo da je plavi kvadrat u sredini slike 15. omeđen nešto svjetlijim obrubom sa svih strana, u kompresiranoj slici to neće biti tako. Povećana komprimirana slika će biti prikazana na slici 19. te će se lako moći uvidjeti promjene koje se događaju prilikom smanjenja slike.



Slika 19 - Uvećana komprimirana slika

	0	1	2
0	87	122	185
1	87	122	185
2	87	122	185
3	87	122	185
4	87	122	185
5	204	72	63
6	204	72	63
7	204	72	63
8	204	72	63
9	204	72	63
10	226	169	24
11	176	228	239
12	176	228	239
13	176	228	239
14	176	228	239

Slika 20 - Vrijednosti srednjeg retka piksela sa umanjene slike 15.

Kako vidimo na slici 19., a sada i na slici 20., imamo izmjenu 4 boje u sredini slike. Očekivano, imamo 3 veća kvadrata i jedan stupac svijetlo plave boje. To se da iščitati i na tablici gdje vidimo da imamo 3 ponavljajuće vrijednosti koje označavaju veće kvadrate te se onda na mjestu 11. pojavi samo jednom kombinacija vrijednosti (176, 288, 239) svijetlo plavoga stupca.

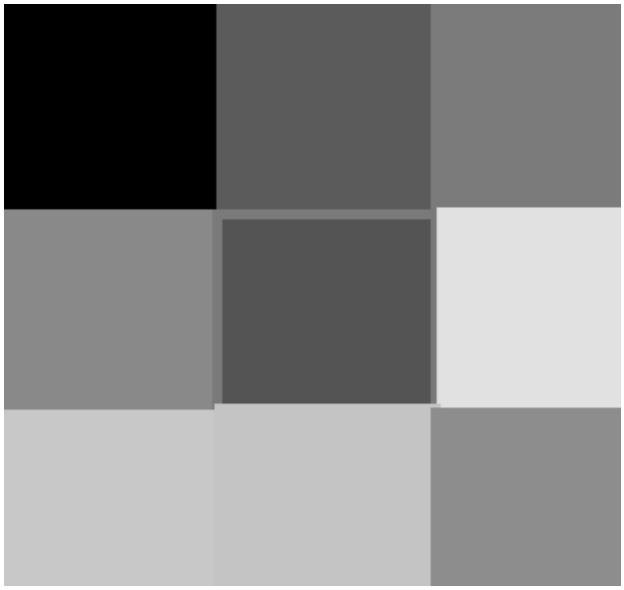
2.4.2. Grayscale

Nakon RGB-a, dolazimo do najbitnijeg sustava boja koji se najviše koristi u računalnom vidu. To je *grayscale,* odnosno slika u crno-bijelim, tj. sivim tonovima. Sustav je tehnički identičan RGB-u, a jedina je razlika to što je za opis slike potrebna jedna matrica koju sačinjavaju vrijednosti od 0 do 255, analogno veličinama kod RGB matrica. *Grayscale* sustav omogućuje redukciju nepotrebnih informacija. Svedemo sliku koja je opisana sa 3 matrice na 1 matricu. Takvom pretvorbom se na slici gube boje, ali ostaju oblici, rubovi i ostale informacije sa kojima je na taj način lakše raditi.

Slika 15. je prikazana u *grayscale* sustavu na slici 21. te će se analogno prethodno pokazanim primjerom pokazati veličine u *Pythonu* za *grayscale* tip slike. Dakle, sliku 15. ćemo transformirati u *grayscale,* sliku ćemo smanjiti 20 puta radi jednostavnosti prikaza intenziteta piksela te prikazati dio cjelokupne matrice slike radi bolje preglednosti. Ovdje je moguće sliku prikazati kao matricu jer je opisana samo jednom matricom, a ne 3 što je slučaj kod RGB-a.

	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	137	137	137	137	137
6	137	137	137	137	137
7	137	137	137	137	137
8	137	137	137	137	137
9	137	137	137	137	137
10	200	200	200	200	200
11	200	200	200	200	200
12	200	200	200	200	200
13	200	200	200	200	200

Slika 21 - Vrijednosti nekih piksela sa slike 22.

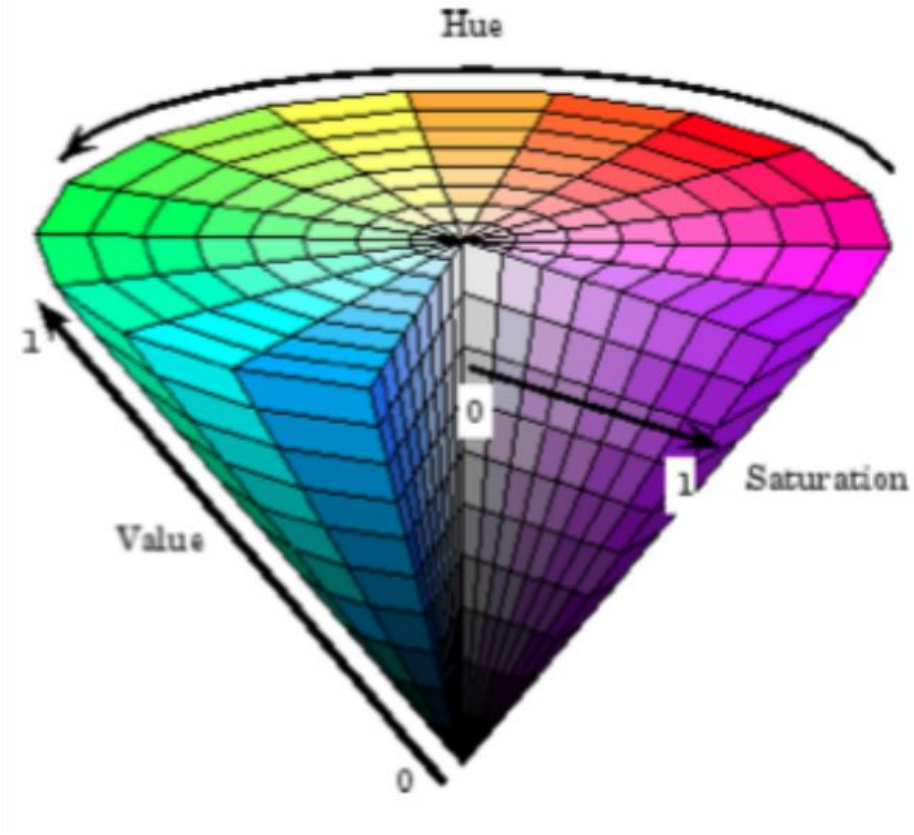


Slika 22 - Slika 15. u grayscale-u

Na slici 21. vidimo prvih 5 stupaca umanjenje slike 22. () Dakle nalazimo 3 različite boje. Crnu i dvije nijanse sive. Crna je, kao što je prethodno već rečeno, opisana kao 0,0,0 u RGB sustavu te je analogno tome opisana sa 0 u *grayscale* sustavu. Preostale dvije nijanse sive su opisane sa 137 i 200. Što je broj veći to je boja svjetlija, kao što se vidi iz priloženog.

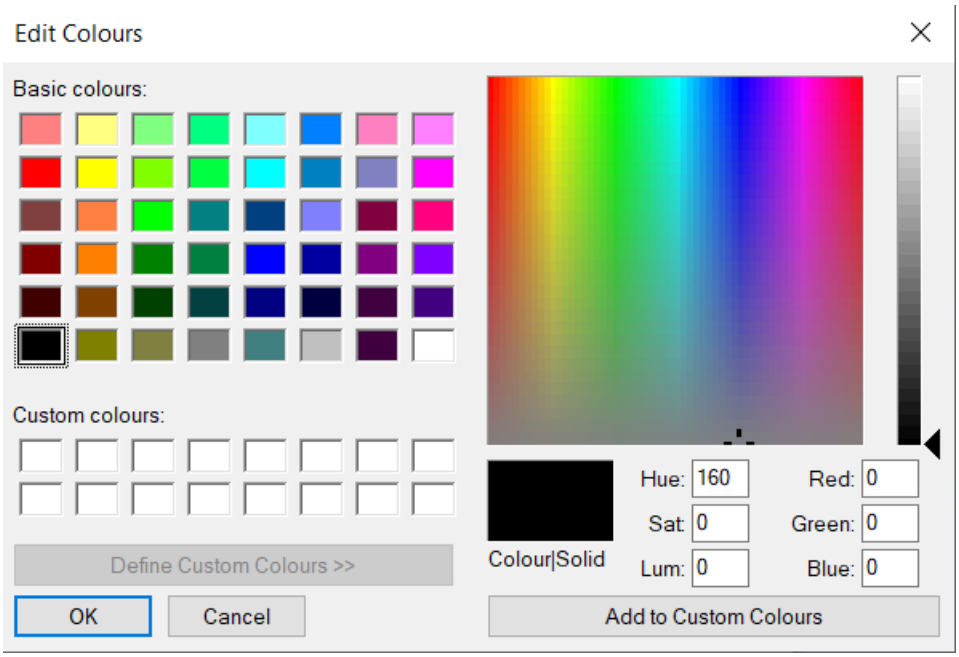
2.4.3. HSV

HSV sustav boja je sustav koji definira boju pomoću 3 veličine: nijansa *(hue),* zasićenje (*saturation)* i vrijednost (*value).* Njegova je prednost nad RGB sustavom to što je sličniji ljudskom percipiranju boja nego RGB sustav. Primjerice, prije ćemo pronaći željenu boju pomoću HSV sustava nego u RGB sustavu. Iz tog razloga se najčešće koristi u grafičkim softverima za kreaciju slika i sličnih sadržaja. Ako se traži određena boja, lako je dalje podešavati nijansu i svjetlinu da se dođe do željenog rezultata. Prikaz HSV-a je dan na slici 23.



Slika 23 - Grafički prikaz HSV sutava boja [4]

Kako je vidljivo sa slike 23. nijansa se definira kutem, tj. može iznositi od 0° do 360°. Iznos od 0° odgovara crvenoj boji, 120° odgovara zelenoj boji, a 240° odgovara plavoj boji. Zasićenje određuje neutralnost odnosno čistoću boje. Što je veće zasićenje to je boja čišća. Zasićenje može iznositi od 0% do 100%, tj. od 0 do 1. Isti raspon veličina poprima i parametar vrijednost. On definira koliko je boja bijela ili crna. Ako je vrijednost parametra 0, boja će biti crna, a ako je vrijednost parametra 1, boja će biti bijela. U programu *Paint* koji dolazi uz operativni sustav *Windows* možemo jednostavno vidjeti kako izgleda paleta boja te biranje boje uz grafički odabir parametara. Dan je prikaz na slici 24. [7]



Slika 24 - HSV paleta boja u programu Paint

Primjene u računalnom vidu su brojne i korisne. Pomoću HSV-a dobijemo takvu definiciju boje gdje se komponente boje ne definiraju intenzitetom boja. To nam omogućava rješavanje određenih problema poput osvijetljenja. Ako imamo sliku neke velike površine koja je u fizičkom svijetu načinjena od jedne boje, na slici to neće biti jednobojno, odnosno računalo to ne može interpretirati kao jednu boju jer će intenziteti boja biti različiti na različitim dijelovima slike zbog drukčijeg osvijetljenja na određenim dijelovima površine. Taj problem se može riješiti pomoću HSV sustava boja, odnosno analizom je moguće vidjeti da se zapravo radi o istoj boji ali drukčijeg osvijetljenja *(value)*.

2.5. OpenCV

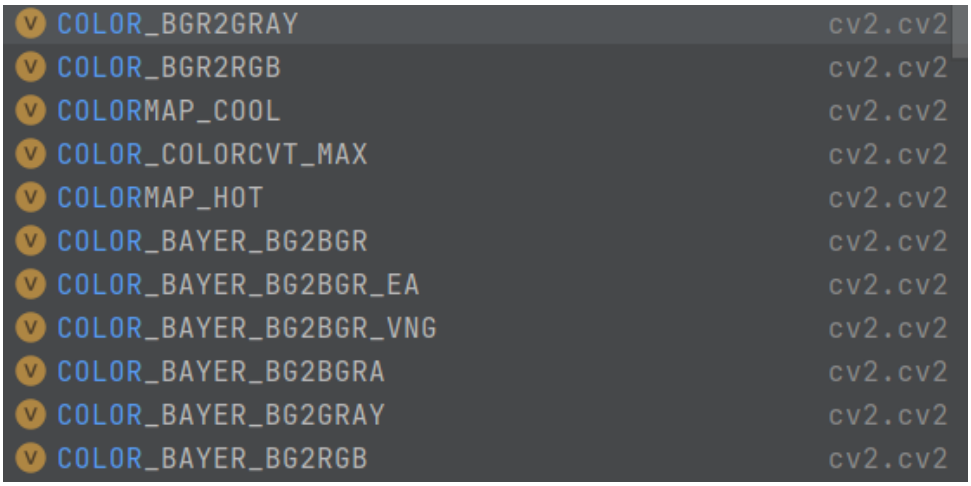
OpenCV je besplatna biblioteka funkcija prilagođenih za računalni vid i strojno učenje. Napravljena je za više programskih jezika, a za potrebe ovog rada se koristi programski jezik *Python* te odgovarajuća OpenCV biblioteka. Sadrži više od 2500 optimiziranih algoritama u koje su uključeni jednostavni i visoko sofisticirani algoritmi. Algoritmi mogu biti korišteni za prepoznavanje lica, identifikaciju objekata, prepoznavanje ljudskih kretnji u videu, praćenje kretnji kamere, praćenje gibajućih objekata itd. Njegove potencijale i mogućnosti su prepoznale najpoznatije korporacije koje ga ujedno i koriste, a neke od njih su Google, Yahoo, Toyota, Sony, IBM, Honda, Microsoft, Intel i mnogi drugi. Neke od primjena za koje se koristi OpenCV – detekcija provala kod video nadzora u Izraelu, praćenje opreme za rudarenje u rudnicima u Kini, pomoć robotima kod navigacije i podizanja objekata, detekcija utapanja u bazenima u Europi, inspekcija etiketa na proizvodima u tvornicama... Moguće ga je koristiti u *C++*, *Python-u*, *Java-i* i *MatLab-u*, a podržan je na operativnim sustavima *Windows, Linux, Android* i *MacOS*. Izvorno je napisan u *C++* programskom jeziku. Biti će opisane neke od osnovnih funkcija iz OpenCV-a koje su potrebne za rješavanje ovog rada. [10]

2.5.1. Promjena sustava boja

Mijenjanje sustava boja je u OpenCV-u jedna od najosnovnijih operacija. Vrlo je jednostavna za izvršiti, a njen rezultat nekada uvelike olakšava daljnji razvoj aplikacije. Najčešća promjena je iz RGB sustava u *grayscale.*

Primjer poziva funkcije: cv.cvtColor(img, cv.COLOR_BGR2GRAY)

Primjer mijenjanja boja je vidljiv na Slika 15. i Slika 22.

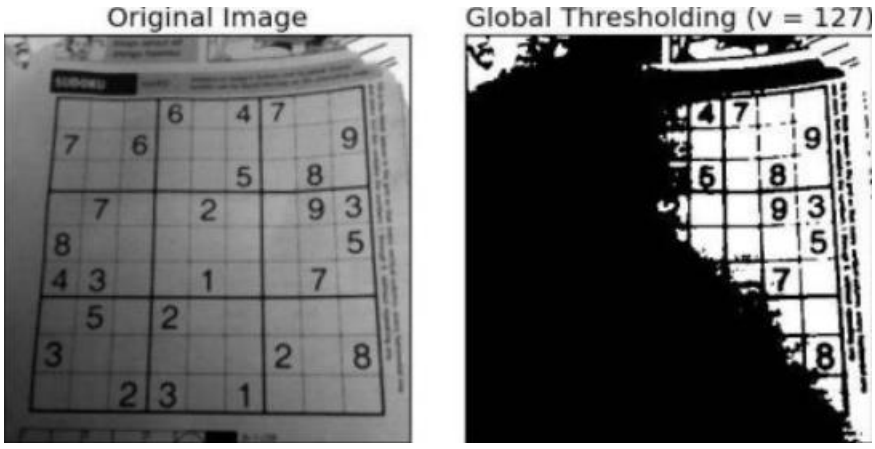


Slika 25 - Neke od moućih promjena sustava boja u OpenCV-u

2.5.2. Threshold

Threshold je funkcija koja omogućuje stvaranje binarne slike s obzirom na intenzitete piksela. S obzirom da na slici postoje pikseli intenziteta od 0-255, moguće je odrediti dva intervala, npr. 0-128 i 129-255 gdje će pikseli čiji su intenziteti unutar određenog intervala imati vrijednost 0 ili 1, zavisno u kojem se intervalu njih intenzitet nalazi. Tako se stvara binarna slika čiji pikseli imaju vrijednosti 0 ili 1. Korištenje ove funkcije je korisno za dobivanje određenih regija na slici čiji pikseli sličnih intenziteta daju neki oblik, predmet ili rubove na slici. Primjer slike na kojoj je primjenjen threshold je dan na Slika 26. Postoji više opcija rada funkcije threshold. To su: binary, binary_inv, trunc, tozero i tozero_inv. Za potrebe ovog rada je korišten binary te je gore objašnjeno funkcioniranje ove opcije rada i dani prikaz na slici je također rezultat ove opcije rada. Ovo su opcije jednostavnog thresholdinga. Postoji i adaptive threshold koji je malo složeniji. On je koristan kod potrebe izvedbe thresholda na slici koja npr. ima promjenjivo osvijetljenje, što je čest slučaj. Tada za dobivanje nekog objekta na slici pomoću običnog thresholda nije moguće korištenje jedinstvenog iznosa granice za threshold jer na slici na drukčijim područjima ima različito osvijetljenje te je nemoguće dobiti njegov oblik na taj način. Algoritam adaptive threshold funkcije uzima manju regiju sa slike, oko piksela, te računa granicu. To omogućuje prolaz po slici, uzimanje promjenjive granice na različitim dijelovima slike kako bi bio omogućen pronalazak određene regije sa slike koja ima različite intenzitete piksela zbog nejednolikog osvijetljenja. [10]

Primjer poziva funkcije: cv.threshold(img, 127, 255, cv.THRESH_BINARY)[1]

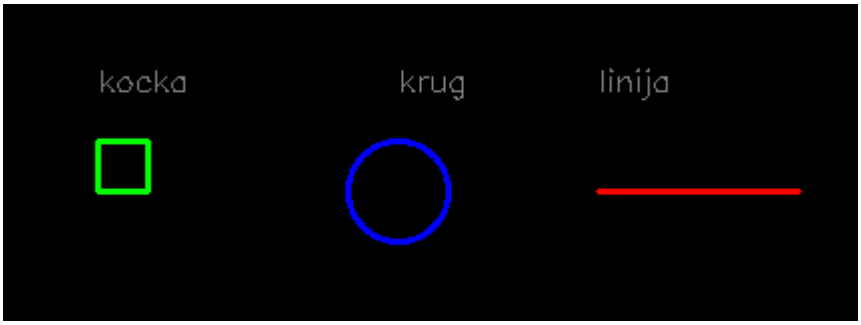


Slika 26 - Primjer thresold funkcije na slici [10]

2.5.3. Crtanje oblika i stavljanje tekstualnih natpisa

Crtanje oblika po slici koja se obrađuje nema važno značenje prilikom računanja nekih parametara. Ono služi za označavanje objekata, regija, područja slike koje su predmet obrade. Oblici se crtaju najčešće kao rubovi željenih regija te se to radi isključivo radi lakše vizualne provjere napravljenog programa. Naravno, neki programi koji zahtijevaju pronalazak objekata na slici, a koje koriste ljudi, moraju označiti dobivene rezultate na slici radi provjere, npr. na proizvodnoj traci u proizvodnji. Osim oblika, čest je slučaj stavljanje natpisa na sliku, odmah pored objekta. Natpisi su potrebni jer ispišu dobivene rezultate koje je onda lako vizualno uočiti te dalje postupati u skladu sa prikazanim informacijama.

Najčešći oblici koji se crtaju na slici su pravokutnik, krug, točka, linije ili kocke. Za njihovo crtanje potrebno je odrediti njihov položaj na slici, veličinu oblika, debljinu linije te boju linije. Analogno njima, za stavljanje teksta na sliku potrebno je definirati položaj teksta na slici, veličinu fonta i tip fonta. Dan je prikaz na slici 27.



Slika 27 - Prikaz crtanja oblika i stavljanja natpisa na sliku

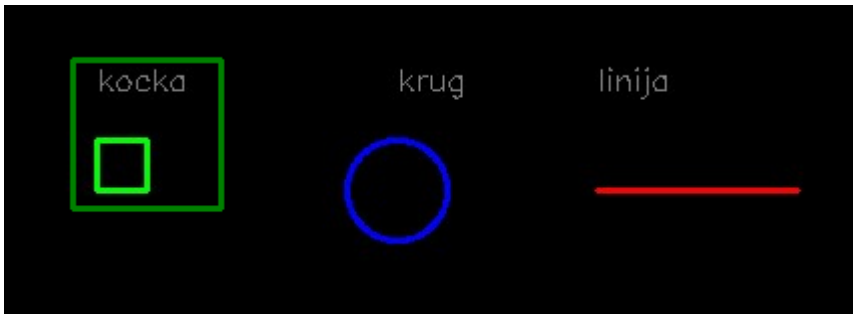
```
cv.rectangle(img, (50, 100), (75, 75), (0, 255, 0), 2)
cv.circle(img, (200, 100), 25, (255, 0, 0), 2)
cv.line(img, (300, 100), (400, 100), (0, 0, 255), 2)

cv.putText(img, 'kocka', (50, 50), cv.FONT_HERSHEY_SIMPLEX, 0.5, (128, 128, 128), 1)
cv.putText(img, 'krug', (200, 50), cv.FONT_HERSHEY_SIMPLEX, 0.5, (128, 128, 128), 1)
cv.putText(img, 'linija', (300, 50), cv.FONT_HERSHEY_SIMPLEX, 0.5, (128, 128, 128), 1)
```

Slika 28 - Primjer koda za dobivanje objekata sa slike 27.

2.5.4. Match template

Match template je jedna od bitnijih funkcija za računalni vid općenito. Ona služi za pronalaženje nekih objekata sa slike koji mogu biti prethodno pohranjeni u bazi podataka. Npr. u tvornici vijaka je potrebno prebrojiti vijke na proizvodnoj traci. Slika vijka je pohranjena na računalu te se ta slika koristi za pronalaženje ostalih vijaka koji se gibaju na traci. Match template funkcija pomoću te slike i njenih karakteristika traži na slici sa kamere objekte koji po svojem obliku, bojom i drugim svojstvima odgovaraju izvornom objektu pohranjenom na slici u računalu. Iskoristit ćemo sliku 27. za pokazivanje rada funkcije match template. Prikaz će biti dan na slici 30. Izrezat ćemo sa slike 27. zeleni kvadrat i natpis 'kocka' (slika 29.) koji će nam predstavljati ono što želimo pronaći na slici. Tada ćemo pomoću funkcije match template pronaći na slici 27. izrezani dio slike i oko pronađene regije nacrtati rub kako bi se vizualno vidjelo ono što je algoritam našao kao podudaranje.

Slika 29 - Dio slike 27. za pronalazak pomoću match template-a



Slika 30 - Rezultat match template-a

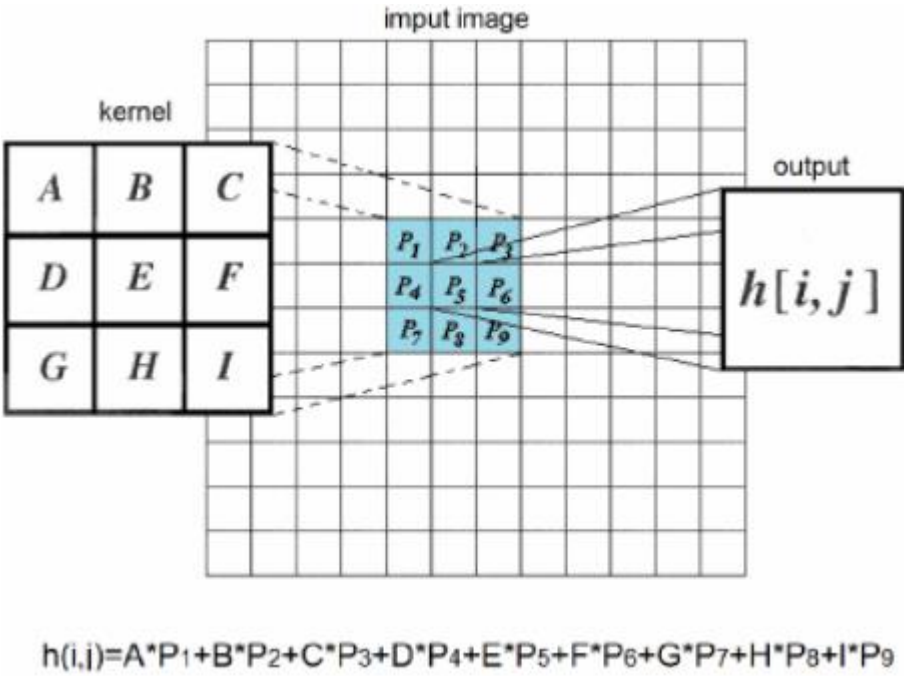
Funkcija ima više algoritama za pronalazak tražene slike. Oni su: TM_CCOEFF, TM_CCOEFF_NORMED, TM_CCORR, TM_CCORR_NORMED, TM_SQDIFF i TM_SQDIFF_NORMED. Svaki algoritam ima svoje načine računanja podudaranja, a najpouzdaniji se pokazao TM_CCOEFF_NORMED koji će biti korišten u ovom radu te je ujedno bio korišten i za ovaj primjer.

Primjer funkcije: cv.matchTemplate(img_find, img, cv.TM_CCOEFF_NORMED)

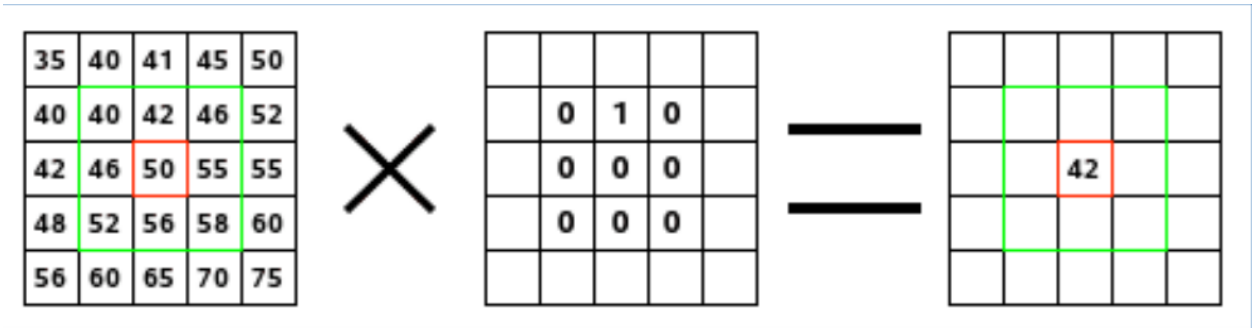
2.5.5. Filteri

Filtriranje je proces obrade slike sa ciljem redukcije šumova, naglašavanjem rubova, zamućenja slike, izoštravanja i slično. Postoji više načina filtracije s obzirom na tip podataka koji obrađujemo. Postoji prostorno filtriranje (matrična 2D slika) i frekvencijsko filtriranje.

Filtriranje u prostornoj domeni je matematičko računanje po elementima matrice. Vrsta filtera je definirana kernelom (matrica svojstvena filteru) koji su obično veličine 3x3 ili 5x5, a rijeđe drugih dimenzija. Svaka pozicija unutar kernela ima svoje značenje, odnosno brojčanu vrijednost koja će se odnositi na konačnu vrijednost izmijenjenog središnjeg piksela. Računanje vrijednosti jednog piksela uzima u obzir intenzitete ostalih piksela koji ga okružuju te se temeljem analize njegove okoline računa vrijednost središnjeg piksela. Filtriranje je proces u kojem kernel prolazi po matrici te mijenja vrijednosti piksela s obzirom na njegovu okolinu. Filtriranjem možemo zamutiti sliku, izoštriti sliku, istaknuti rubove, izdvojiti rubove i sl. Prikaz procesa filtriranja je pokazan na slikama ispod.

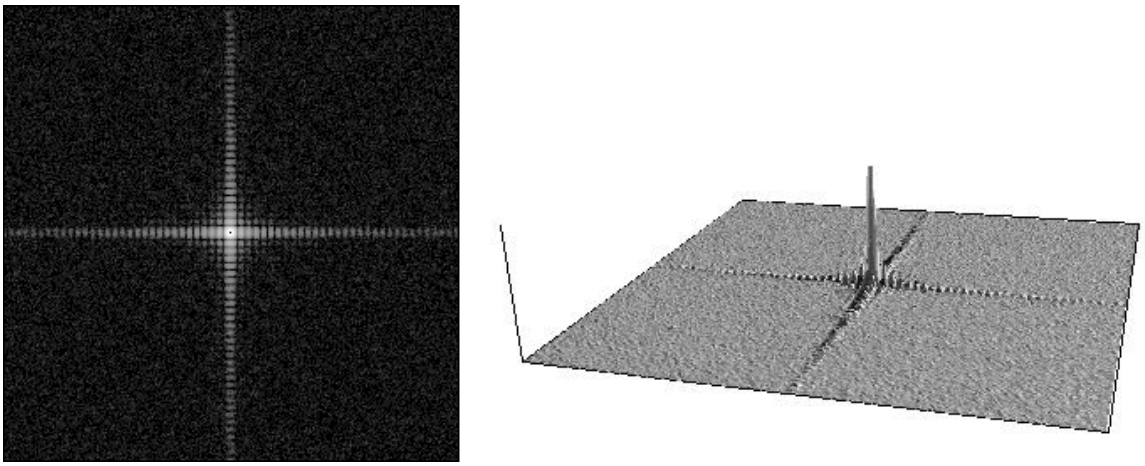


Slika 31 – Kernel, slika i konačni piksel [4]



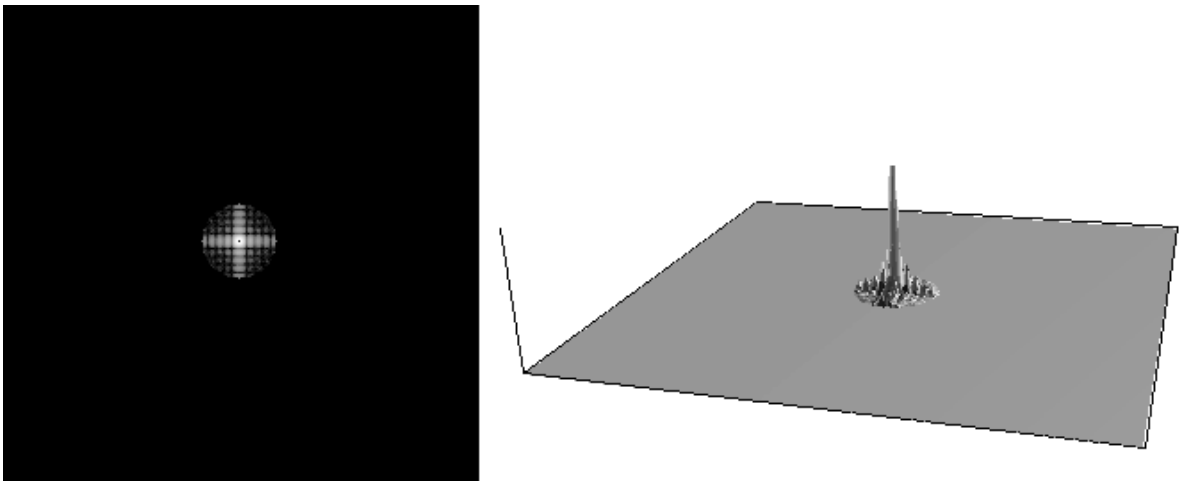
Slika 32 - Primjer dobivanja vrijednosti piksela [4]

Za filtriranje u frekvencijskoj domeni je potrebno provesti kroz sliku Fourierovu transformaciju koja transformira sliku iz prostorne u frekvencijsku domenu. U frekvencijskoj domeni se slike filtriraju prolaskom filtera koji uklanjaju niske ili visoke frekvencije. Prolaskom filtera niskih frekvencija slika postaje mutnija, a prolaskom filtera visokih frekvencija slika postaje oštrija. Primjena filtriranja u frekvencijskoj domeni ima prednost nad prostornom domenom jer je brže filtrirati sliku na ovaj način nego prolaziti kroz cijelu prostornu domenu slike te računati vrijednosti za svaki piksel zasebno. Najpoznatiji filter iz frekvencijske domene je *Gaussian blur* koji čini sliku mutnijom, a to znači da je on filter niskih frekvencija te uklanja područja visokih frekvencija i smanjuje oštrinu slike. Na slici 33. vidimo primjer frekvencijske domene slike.

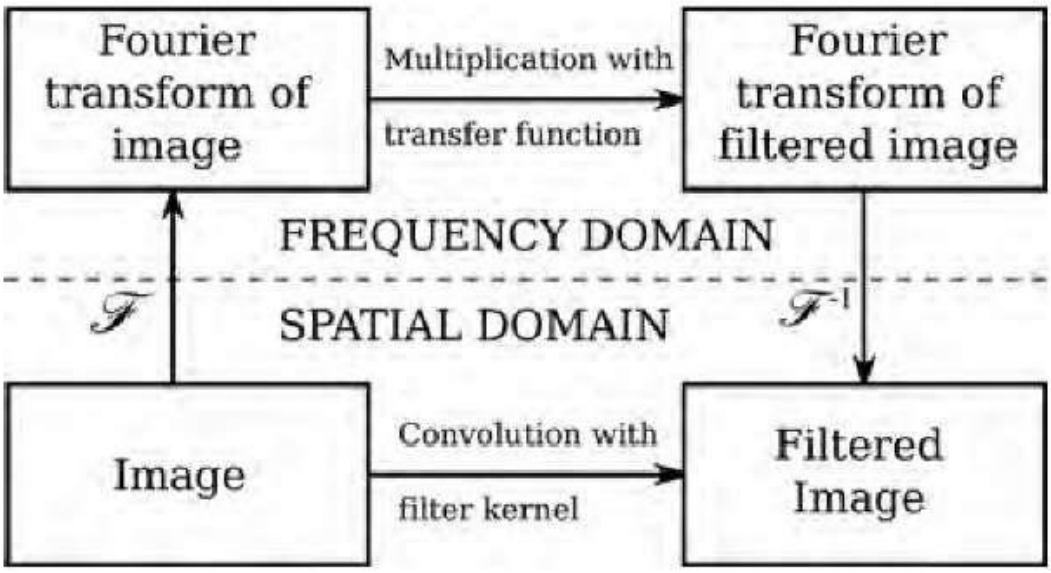


Slika 33 - Frekvencijska domena slike[11]

Na slici 34. će biti prikazana slika 33. sa uklonjenim niskim frekvencijama. Nakon izvršavanja filtriranja slika se vraća iz frekvencijske domene u prostornu obrnutom Fourierovom transformacijom.

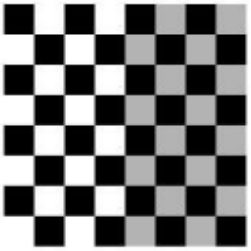
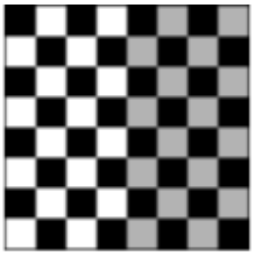


Slika 34 - Rezultat filtriranja filterom visokih frekvencija slike 33.[11]

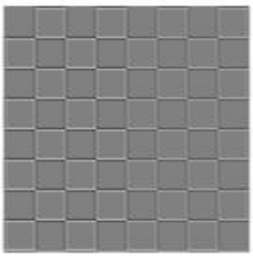
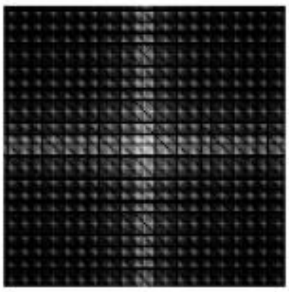
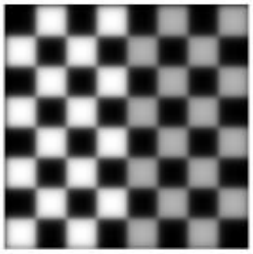


Slika 35 - Shema filtriranja u prostornoj i frekvencijskoj domeni [13]

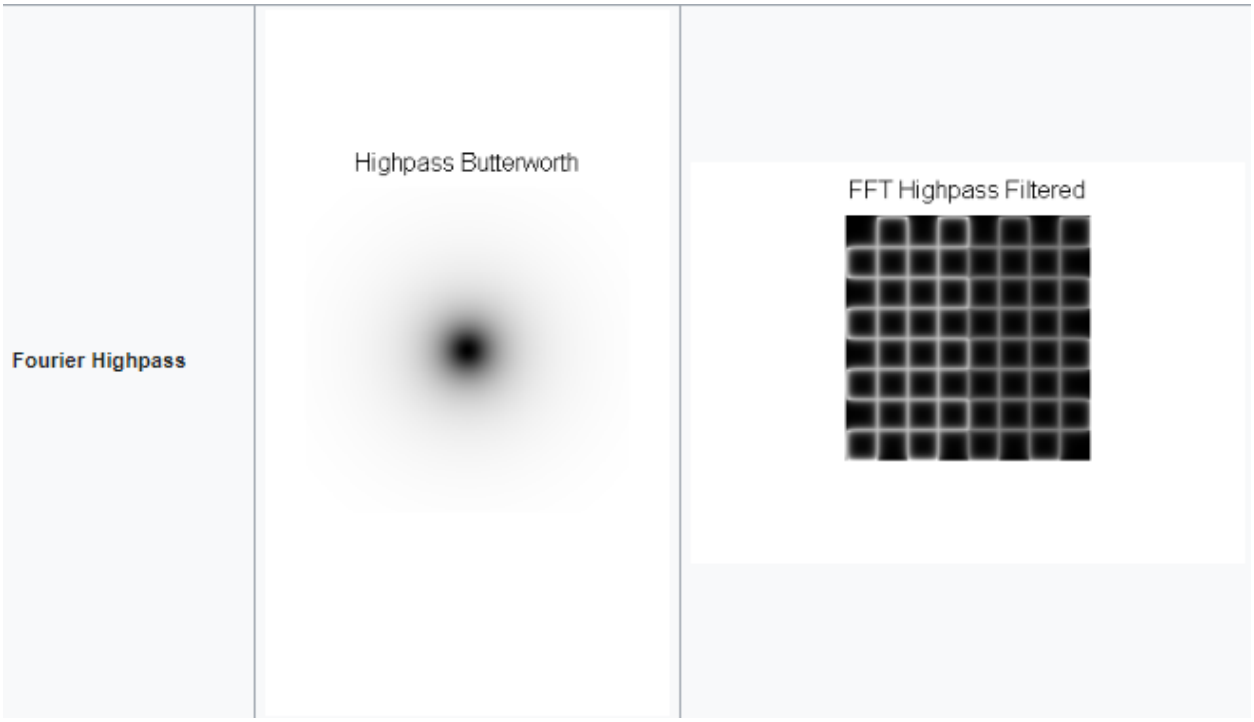
Dalje će biti prikazana slika i na njoj će se iskoristiti neki filteri te će ti rezultati biti prikazani ispod kako bi se dao uvid u mogućnosti i rezultati tih filtera.

Filter type	Kernel or mask	Example
Original Image	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	Identity (Original) 
Spatial Lowpass	$\frac{1}{9} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	3 × 3 Mean Blur 

Slika 36 - Filteri 1/3 [12]

Spatial Highpass	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	Laplacian Edge Detection 
Fourier Representation	Pseudo-code: image = checkerboard F = Fourier Transform of image Show Image: log(1+Absolute Value(F))	FFT Representation 
Fourier Lowpass	Lowpass Butterworth 	FFT Lowpass Filtered 

Slika 37 - Filteri 2/3 [12]

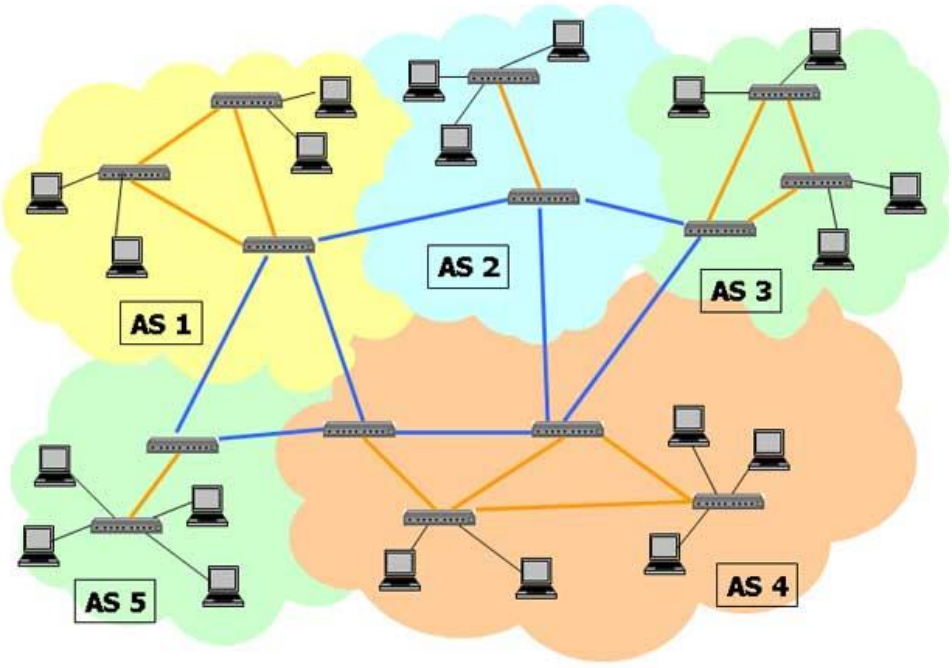


Slika 38 - Filteri 3/3 [12]

3. Internet, mrežni protokoli i standardi

Početak interneta se pojavljuje 60-ih godina prošlog stoljeća. Razvija se mreža za povezivanje računala sa ciljem brze i sigurne komunikacije. Davanje usluge interneta široj populaciji pruža bezbrojne mogućnosti za njegovu uporabu. Razvija se infrastruktura za globalnu povezanost računala, razvijaju se mrežni protokoli za specifične prijenose podataka, razvijaju se programske podrške i aplikacije, web stranice, e-mail... Sve veća zainteresiranost populacije za korištenjem interneta pruža šansu IT sektoru za nove proizvode baziranim na internetskim uslugama. Danas je internet postao glavno sredstvo prijenosa podataka, objavljivanja informacija, širenja vijesti, komunikacije i skladištenja podataka. Internet je temelj industrije 4.0 zbog omogućavanja povezanosti uređaja, senzora, računala i ljudi u jedan sustav.

Internet je skupina povezanih mreža strukturiranih tako da tvore globalnu mrežu koja omogućuje brzi prijenos informacija između određenih točaka mreže. Temeljnu strukturu interneta stvaraju tvrtke koje posjeduju internetsku infrastrukturu preko koje putuju podaci među državama ili kontinentima. Ispod njih se u hijerarhiji nalaze nacionalne i regionalne kompanije koje provode infrastrukturu na području države ili regije. Te tvrtke omogućuju kompanijama, domovima i drugim organizacijama pristup internetu putem njihove infrastrukture. One su spojene na globalnu infrastrukturu te se time omogućuje povezanost globalne internetske mreže. Na kraju hijerarhije se nalazi lokalna jedinica (npr. kuća) koja se sastoji od računala i ostalih uređaja koji se spajaju na internet. Ti se uređaji spajaju na *router,* uređaj kojeg postavlja pružatelj interneta te omogućuje krajnjim uređajima nekog klijenta da koriste internetsku vezu. U takvoj lokalnoj jedinici je također moguće uspostaviti LAN (*local area network)* vezu u kojoj su međusobno povezani uređaji preko čvora (*router-a*). Takva mreža je vidljiva samo lokalno spojenim uređajima, odnosno nevidljiva je ostalim korisnicima interneta koji se ne nalaze na tom *router-u.* Svaki uređaj koji je spojen na internet posjeduje svoju adresu. Ta se adresa zove IP adresa. IP adrese služe kao identifikatori prilikom slanja i primanja podataka pomoću kojih čvorovi u mreži znaju kome treba proslijediti dobivene podatke kako bi došli na potrebno odredište u što kraćem vremenskom roku. IP adresa se sastoji od 4 broja, npr. 192.168.2.0. Svaka regija posjeduje svoje karakteristične IP adrese te se prema tome može odrediti gdje se nalazi koje računalo. To se koristi za preusmjeravanje podataka u čvorovima mreže prilikom slanja kako bi njihov put što kraći i brži.



Slika 39 – Pojednostavljena shema strukture interneta [14]

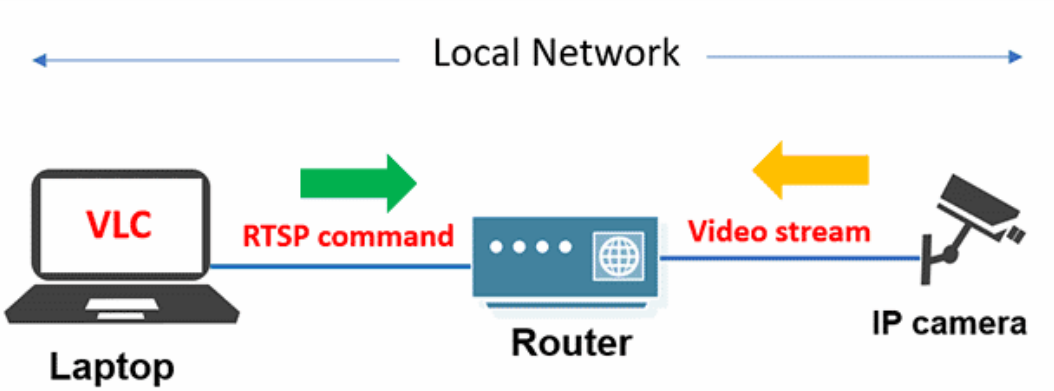
Za prijenos podataka su razvijeni mnogobrojni protokoli, zavisno o tipu i primjeni podataka. Najpoznatiji protokol danas je HTTP protokol koji se koristi za prikaz web stranica u internetskim preglednicima. Osim njega, poznati su još TCP/IP, UDP, FTP, SMTP, POP.... Broj protokola je mnogobrojan, kao i pripadajući tipovi podatka koji se šalju. Svaki tip podataka ima svoju krajnju primjenu za koju se koristi te se iz toga razloga koriste brojni protokoli zbog sigurnosti te kako bi se podaci u mreži lakše razlikovali jedni od drugih. Za potrebe ovog rada će biti potrebno raditi sa RTSP, SOAP i WSDL protokolima unutar ONVIF standarda.

3.1. RTSP

RTSP (*real-time streaming protocol)* je protokol koji služi za prikazivanje video sadržaja. To je bio najkorišteniji alat za prikazivanje videa na web stranicama, ali u zadnje vrijeme razvojem HTML5 jezika, razvija se noviji i bolji RTMP protokol koji će preuzeti ulogu prikazivanja videa na webu. Osim toga, RTSP protokol je i dalje jedan od najpoželjnijih protokola koji se koriste kod IP kamera.

Ovaj je protokol razvijen 1996. godine te je omogućavao korisnicima da slušaju glazbu ili gledaju video sadržaje bez potrebe za prethodnim skidanjem datoteka. RTSP protokol se ponaša kao upravljač mreže. Kada korisnik pokrene zahtjev za gledanjem videa, RTSP protokol šalje naredbu medijskom serveru te se uspostavlja veza između medijskog servera i uređaja na kojem se prikazuje sadržaj. RTSP protokol posjeduje nekoliko zahtjeva,

odnosno funkcija kojima je moguće upravljati sa medijskim sadržajem, odnosno vezom između servera i računala. Neke od funkcija su SETUP (uspostavljanje veze sa serverom), PLAY (pokretanje videa), PAUSE (zaustavljanje videa), REDIRECT (spajanje klijenta na novi server), itd. U sklopu ovog rada RTSP protokol će biti korišten za kontinuirano dobivanje slike sa kamere. Za spajanje na kameru će se koristiti OpenCV modul koji u sebi ima implementiranu funkciju za dobivanje slike sa različitih izvora, a sam proces dobivanja slike će se voditi putem RTSP protokola. Podaci koji putu od strane servera prema računalu se šalju TCP protokolom. [16]

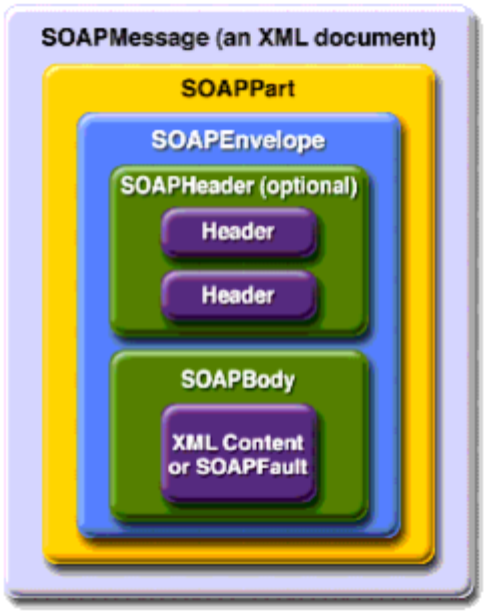


Slika 40 - Shema rada sustava sa RTSP protokolom

3.2. SOAP

SOAP (*simple object access protocol*) je protokol čiji je cilj izmjena strukturiranih podataka između računala u mreži. Poruke se šalju putem HTTP protokola, a poruke su u XML formatu. S obzirom da se poruke šalju preko HTTP protokola koji je svojstven web servisima, SOAP protokol omogućuje slanje poruka i informacija web servisima putem mreže te pozivanje nekih funkcija na udaljenim web servisima, odnosno udaljenim računalima ili nekim drugim uređajima, u ovom slučaju IP kamere.

Podatci koji se šalju SOAP protokolom su upakirani i strukturirani na specifičan način. Podatak koji se šalje sadrži više slojeva od kojih svaki sloj nosi određene informacije potrebne krajnjem udaljenom uređaju (Slika 41.). Kada krajnje računalo primi poruku, odmotavanjem se dobivaju sve potrebne informacije ili postavke koje je potrebno definirati za npr. pozivanje neke funkcije na udaljenom uređaju. Takva poruka sadržava korisnička imena, šifre, korištene protokole, varijable te funkciju koju želimo pozvati na udaljenom računalu i dr. [17]



Slika 41 - Skica SOAP podatka [17]

3.3. WSDL

WSDL (*web services definition language*) je skup pravila koji opisuju kako se određeni web servisi mogu koristiti, koje poruke im se mogu slati, kako moraju biti strukturirane poruke i informacije koje se šalju. WSDL je napisan u XML formatu te se u njemu nalaze opisane sve funkcionalnosti nekog web servisa sa svim potrebnim opisima. U ovakvim tipovima protokola i standarda se najčešće koristi XML format jer je ljudski čitljiv i razumljiv tip zapisa podataka, a ujedno ga i računalo razumije. U ovom radu će se koristiti WSDL u sklopu ONVIF standarda. Kada se skine ONVIF standard moguće je naći WSDL datoteku u pripadajućoj mapi na računalu. U toj datoteci se nalazi popis svih funkcionalnosti koje je moguće izvršiti na uređajima koji rade po ONVIF standardu. [17]

```
<xs:element name="DateTimeType" type="tt:SetDateTimeType">
    <xs:annotation>
        <xs:documentation>Defines if the date and time is set via NTP or manually.</xs:documentation>
    </xs:annotation>
</xs:element>
```

Slika 42 - Dio WSDL dokumenta zapisan u XML formatu

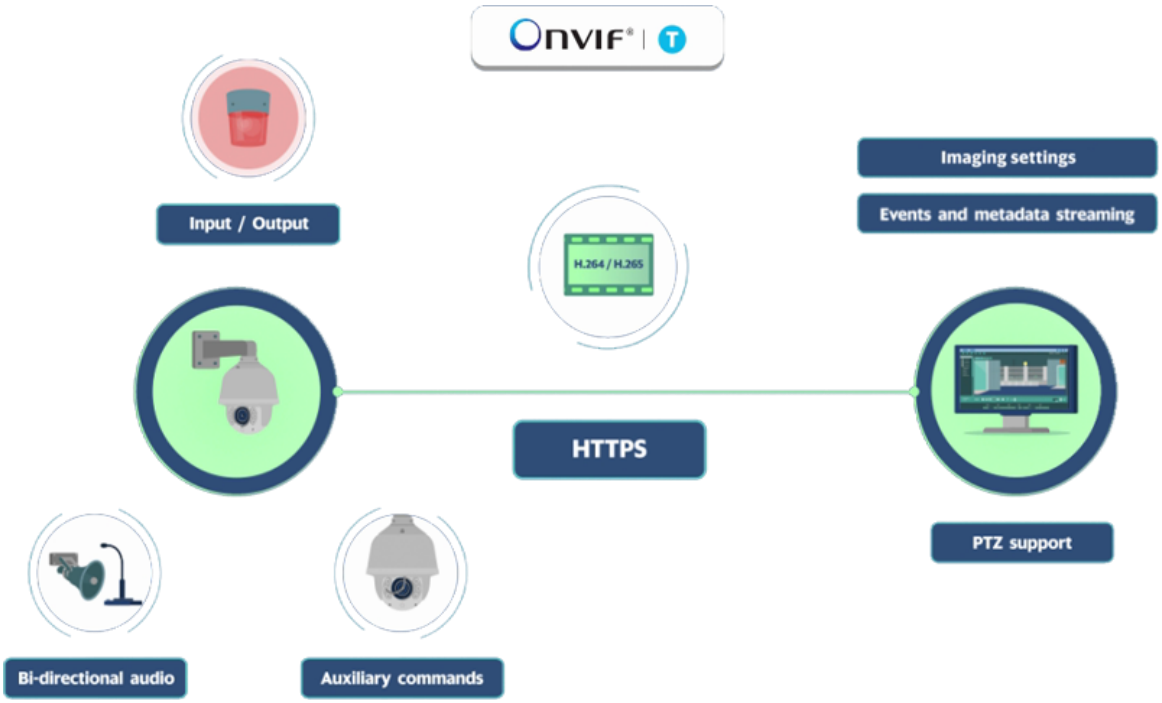
3.4. ONVIF

ONVIF (*open network video interface forum*) je standard koji koristi razne tehnologije upakirane u sebe kako bi se lako radilo sa fizičkim video sustavima. Razvijen je sa ciljem standardiziranja komunikacije IP proizvoda unutar video nadzornih sustava sa ostalim fizičkim dijelovima sustava, npr. računala. ONVIF je organizacija koju su 2008. pokrenule korporacije Bosch Security Systems, Sony te Axis Communications. [18]

ONVIF želi standardizirati mrežno sučelje mrežnih video proizvoda. Definira okvir mrežne video komunikacije pomoću određenih protokola i web servisa te uključuje IP i sigurnosne zahtjeve.

Osnovne funkcionalnosti

- IP konfiguracija
- Detekcija uređaja
- Upravljanje uređajima
- Konfiguracija medijskih sadržaja
- Real-time praćenje video sadržaja
- Rukovanje događajima
- PTZ kontrole
- Video analitika
- Sigurnost



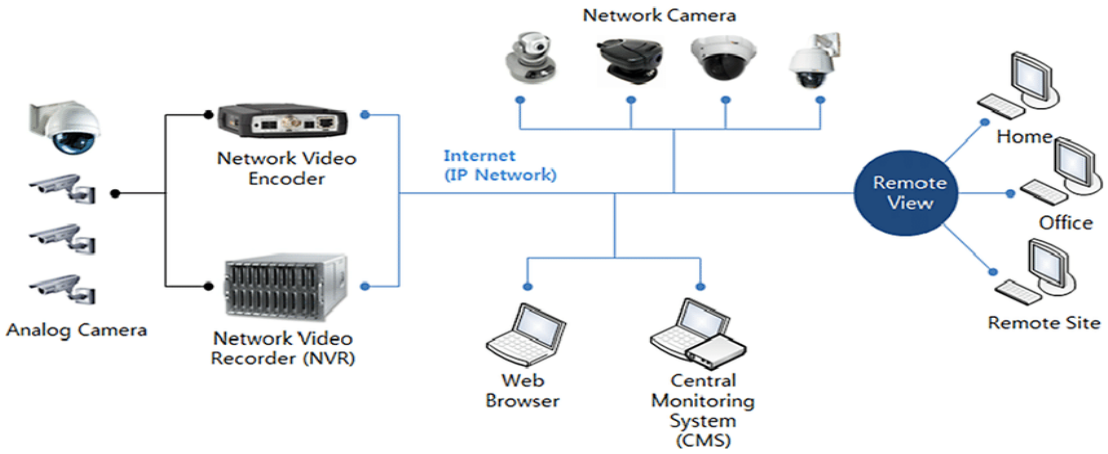
Slika 43 – Shema primjera video sustava kompatibilnog sa ONVIF standardnom [18]

U ovom radu će se koristiti ONVIF standard za spajanje na IP kameru te njeno pokretanje. ONVIF modul je besplatan te ga je moguće skinuti za *Python* programski jezik te će se u sklopu njega pisati kodovi za manipulaciju kamere.

4. Nadzorne kamere

Kamera je proizvod čija je svrha zabilježiti ili nadgledati neki dio prostora. U početku su kamere služile za stvaranje video zapisa i obilježavanje nekih događaja. Razvojem tehnologije i društvenih potreba javila se potreba za kreiranjem kamera koje će nadgledati prostore ili procese u svrhe sigurnosti i kontrole. Razvijaju se kamere koje su spojene na udaljeni kontrolni sustav gdje se pohranjuju zabilježeni video zapisi. Razvijaju se IP kamere koje je moguće kontrolirati i bežičnim putem, preko internetske veze. Takve kamere se kontroliraju prethodno navedenim protokolima i standardima. Nerijetko, kamere u sebi imaju ugrađene mehaničke sustave koji im omogućuju okretanje oko osi. Takve kamere se zovu IP PTZ kamere. PTZ (*pan-tilt-zoom)* je oznaka za kameru kod koje je moguće rotiranje oko dvije osi te posjeduje mogućnost povećanja/smanjenja slike. Kada se sa računala pošalju na web servis kamere odgovarajuće informacije i podaci, moguće je kontrolirati kamerom na mnogobrojne načine ili dobivati povratne informacije. Po tom principu rade nadzorni video sustavi gdje je stalna komunikacija između kamere i središnje upravljačke jedinice, odnosno računala.

Nadzorni video sustavi su strukturirani tako da su kamere spojene žičano ili bežično na mrežu, u slučaju bežičnog načina rada mora postojati lokalni *router* koji daje mogućnosti WiFi-ja putem kojeg će se računalo ili kamera spojiti na mrežu bez žice. Kamere se postavljaju na željene lokacije te se njima dalje upravlja putem računala. Na računalu mora biti potreban softver kompatibilan sa cijelim sustavom koji tada može slati ili primati informacije i podatke sa kamere putem mreže.



Slika 44 - Shema nadzornog video sustava

5. Python

Python je ime programskog jezika koji je danas jedan od najpopularnijih programskih jezika zbog svoje jednostavnosti korištenja, razumijevanja i široke primjene. Besplatan je te stoga ima veliku bazu korisnika koji stalno unaprijeđuju funkcionalnosti koje pruža. Za njega su napisani mnogi moduli koji mu pružaju lakoću primjene u mnogobrojnim područjima programiranja. Moduli su skupine napisanih funkcija za određenu primjenu gdje je za izvršavanje određenog zadatka dovoljno pozvati funkciju iz modula umjesto pisanja cijelog koda za izvršavanje određenog zadatka. To je najveća prednost ovog programskog jezika. Svakim danom je njegov opseg funkcionalnosti sve veći što privlači mnogobrojne programere da krenu učiti ovaj jezik. Koristi se za kreiranje web servera, matematičke algoritme, računalni vid, umjetnu inteligenciju, mrežnu komunikaciju, robotiku i dr. S obzirom na tako širok spektar mogućnosti, on je bio korišten za izradu ovog zadatka. Napravljeni su moduli koji mu omogućuju lako spajanje na IP kamere, dobivanje slike sa kamere, obradu slike dobivene sa kamere, računanja podataka temeljenih na analizi slika te manipulaciju kamere u odnosu na izračunate podatke. Riječ je već bila u ranijim poglavljima o ONVIF standardu te *OpenCV*-u koji su korišteni za izradu zadatka.

S obzirom da je *Python* programski jezik, a njegovo izvorno sučelje za razvoj programa pojednostavljeno, korišteno je razvojno grafičko sučelje, odnosno IDE (*integration development environment) PyCharm*. On pruža mnogobrojne mogućnosti jer je razvijen ponajprije za *Python* te je iz toga razloga jedan od vodećih IDE-eva za razvoj programa pisanih u *Pythonu.* S obzirom na široki spektar primjene ovog programskog jezika, *PyCharm* pokriva veliki dio onoga što je potrebno za rad sa programskim jezikom.

Python je skriptni jezik što znači da se izvršavaju linije koda jedna iza druge, onako kako je kod napisan, za razliku od npr. C jezika gdje se cijeli kod pretvara u računalno razumljiv jezik, a potom izvršava. Izvorno je pisan u C jeziku. [19]

Slika 45 – Logo programskog jezika Python [19]

6. Rad

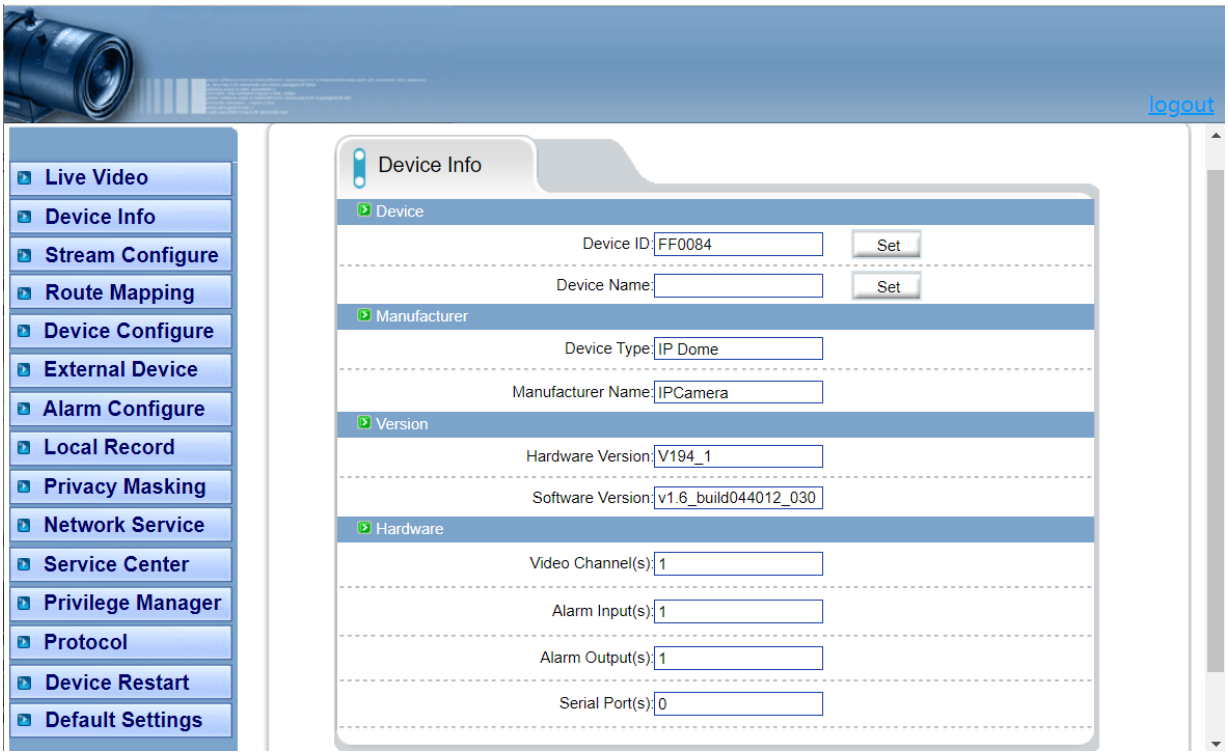
6.1. Korištena oprema

6.1.1. IP PTZ kamera

Za potrebe ovog zadatka bilo je potrebno koristiti IP PTZ kameru. U laboratoriju se nalazi IP PTZ Sunell kamera. Kao što naziv kaže, kamera ima mogućnost rotacije oko dvije osi te povećanja slike. *Pan* označava rotaciju oko y osi za 360°. *Tilt* označava rotaciju oko osi x, te je ovdje moguće rotirati za 90. *Zoom* označava povećanje/zumiranje slike te je kamera u mogućnosti sliku povećati 20 puta. Rezolucija slike je 704x576 px. Broj slika u jednoj sekundi se mjeri parametrom FPS (*frames per second).* Kamera radi sa 25 FPS-a, što znači da se u jednoj sekundi promjeni 25 slika. Što je više slika u jednoj sekundi izmijenjeno, to je bolja i uglađenija kvaliteta videa. Kamera posjeduje infracrvena svijetla te prikaz slike u infracrvenom spektru boja. To omogućuje kameri rad po noći sa dobrom vidljivošću. Kamera je prikazana Slika 46.

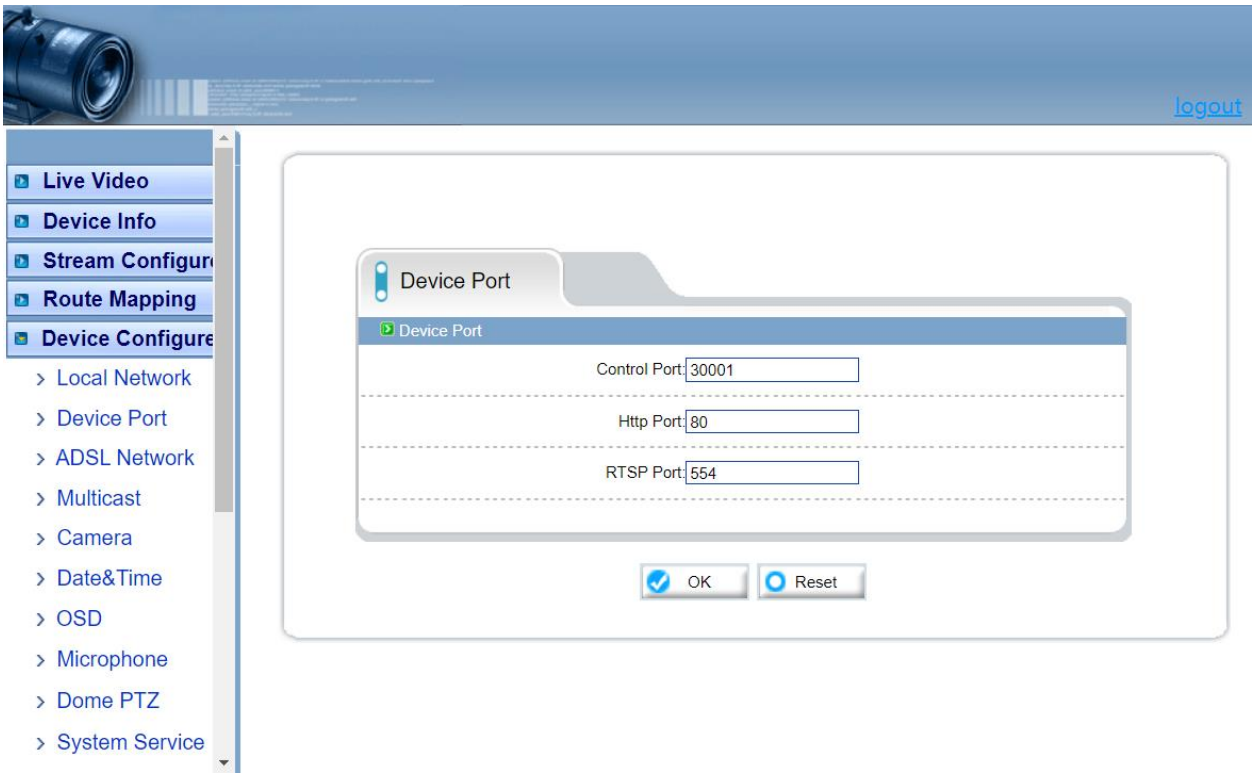
Slika 46 - IP PTZ Sunell kamera

Kamera posjeduje svoje web sučelje preko kojega je moguće pregledati i uređivati postavke kamere. Na to web sučelje se spaja putem internetskog preglednika. Web sučelje je prikazano na Slika 47.



Slika 47 - Web sučelje kamere

Na kameru se spaja putem odgovarajuće IP adrese. Kamera za promet podataka koristi 3 različita *port-a.* Jedan je *port* 80 koji služi za prijenost podataka putem HTTP protokola. Drugi *port* je 30001 te se putem njega kameri šalju razne naredne, npr. naredba da se kamera okreće u desnu stranu. Zadnji *port* je 554. Putem njega kamera šalje sliku u realnom vremenu preko RTSP protokola. Dakle, kada je poželjno dohvatiti sliku sa kamere potrebno je komunicirati sa kamerom preko navedenog porta. Postavke navednih *port-ova* su prikazani na Slika 48.

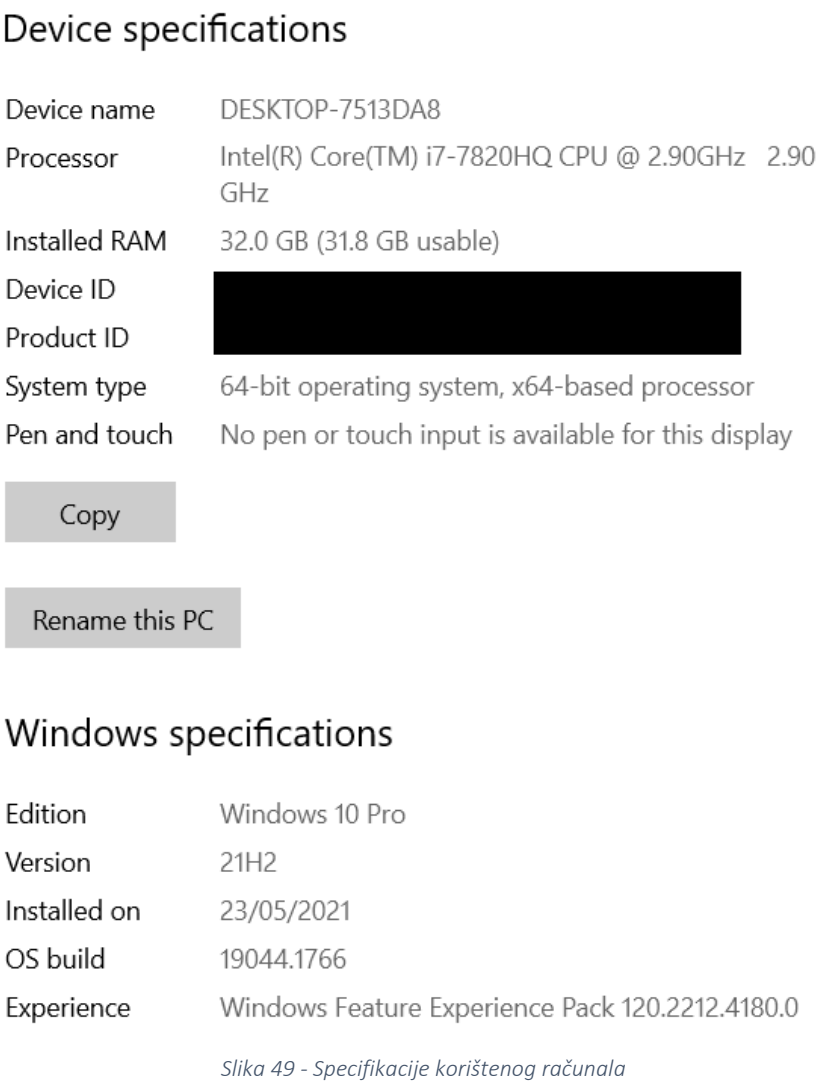


Slika 48 - Port-ovi koje koristi IP PTZ kamera

Kameru se preko ethernet kabela spaja na lokalnu mrežu. Kada je kamera spojena na lokalnu mrežu, moguće joj je pristupati na navedeni način i kontrolirati ju. Osim preko web sučelja, kamerom je bilo pristupano putem *ONVIF Device Manager* programskog paketa. To je softver razvijen od strane razvijatelja ONVIF standarda te je stoga bilo vjerodostojno za isprobati funkcionalnosti kamere. Osim toga, kameri se pristupalo putem pisanih programskih kodova korištenjem ONVIF standarda nakon što su se isprobale prethodno navedene metode.

6.1.2. Prijenosno računalo Lenovo Thinkpad P51

Za potrebe pisanja programskih kodova korišteno je računalo *Lenovo Thinkpad P51*. Posjeduje Windows 10 operativni sustav. Za rješavanje zadatka je bilo potrebno instalirati već ranije navedeni programski jezik *Python 3.10.2* te pripadajući IDE *PyCharm Community Edition 2021.3.3.* Specifikacije računala su prikazana na slici ispod.



Slika 49 - Specifikacije korištenog računala

6.2. Rješenje zadatka

U radu je bilo potrebno napraviti programski kod pomoću kojega bi kamera pratila promjene u sceni te ukoliko se neki objekti na slici miču, kamera ih je potrebna pratiti i držati u fokusu, odnosno centrirati ih u slici. Za rješenje takvog zadatka bilo je potrebno riješiti nekoliko problema. Prvo je bilo potrebno dobiti sliku sa kamere unutar programskog jezika *Python.* Nakon toga je bilo potrebno pomoću ONVIF standarda, unutar *Python-a*, postići upravljivost kamere. Nakon što se postigla upravljivost kamere pomoću pisanih funkcija unutar *Python-a*, bilo je potrebno putem prethodno dobivene slike sa kamere analizirati dobivenu sliku te napraviti programski kod koji bi detektirao promjene na sceni. Nakon detekcije promjena na sceni, potrebno je osmisliti način kojim bi kamera pratila promjenu na sceni. U razmatranju je bilo više pristupa, a odabrani pristup će biti detaljnije prikazan u idućim poglavljima. Dakle, za pratnju gibajućih objekata u sceni je bilo potrebno riješiti problem globalnog i lokalnog koordinatnog sustava kako bi se mogao odrediti položaj gibajućeg objekta u prostoru i na slici kamere. Nakon što se riješio taj problem, bilo je potrebno povezati gibanje objekta u sceni te na temelju njegovog položaja gibati kameru kako bi ga kamera na kraju njegovog gibanja i dalje imala u kadru.

Ukratko, bilo je potrebno riješiti slijedeće probleme:

- Spajanje na kameru
- Detektirati gibanje na slici
- Navesti kameru da prati gibajući objekt

Naravno, prilikom rješavanja ovih problema, javilo se mnoštvo manjih problema. Detaljan opis, funkcionalnost programa te suočeni i rješeni problemi će biti opisani u idućim poglavljima.

6.2.1. Spajanje na kameru

6.2.1.1. Dobivanje slike sa kamere

Nakon što se kamera spoji u električnu struju i ethernet kabelom na lokalni *router*, odnosno u lokalnu mrežu, kamera je spremna za korištenje. Da bi se spojilo na kameru putem programskog jezika *Python* bilo je potrebno skinuti 2 modula. Jedan modul je korišten za upravljanje kamerom *onvif-py3*, a drugi modul je korišten za dobivanje i obradu slike sa kamere, odnosno *opencv-contrib-python*. Navedeni modul je proširenje standardnog modula *opencv* te posjeduje neke dodatne funkcionalnosti.

Dakle, prvo se pri rješavanju zadatka htjelo dobiti sliku sa kamere. To se postiglo pomoću *OpenCV* modula. Za spajanje na kameru se koristila funkcija kojom se inicijalizira spajanje računala i kamere. S obzirom da je video sklop mnogobrojnih slika koje se izmjene u kratkom vremenskom periodu, bilo je potrebno napraviti petlju kojom bi se osigurao kontinuiran i neprekidan prikaz slika. Unutar te petlje je potrebno definirati da se u svakoj iteraciji petlje dohvati nova, trenutna slika sa kamere te da se istu prikaže kako bi se mogla pratiti slika sa kamere. Dakle, za spajanje na kameru, dohvaćanje i prikaz videa, odnosno slika u realnom vremenu, potrebno je znati IP adresu sa koje se preuzima slika putem RTSP protokola. Navedena adresa glasi: rtsp://192.168.0.200:554/snl/live/1/1/Ux/sido=-Ux/sido=.

Sa navedene adrese možemo uočiti 3 stvari o kojima je već ranije bila riječ. Možemo uočiti da poveznica na početku sadrži *'rtsp'* što upućuje na protokol kojim se prometuje podacima. Druga stvar je IP adresa same kamere, odnosno 192.168.0.200. Treća stvar koja se može uočiti je *port* preko kojega kamera šalje sliku. U ranijem poglavlju je bio prikaz da se na *port-u* 554 odvija promet RTSP protokolom. On se na poveznici nalazi iza same IP adrese. Ispod će biti prikazan kod kojim se dobiva slika sa kamere, a na slici ispod koda će biti prikazan prozor koji se otvara unutar *Python-a* kada se želi prikazati slika, odnosno video sa kamere.

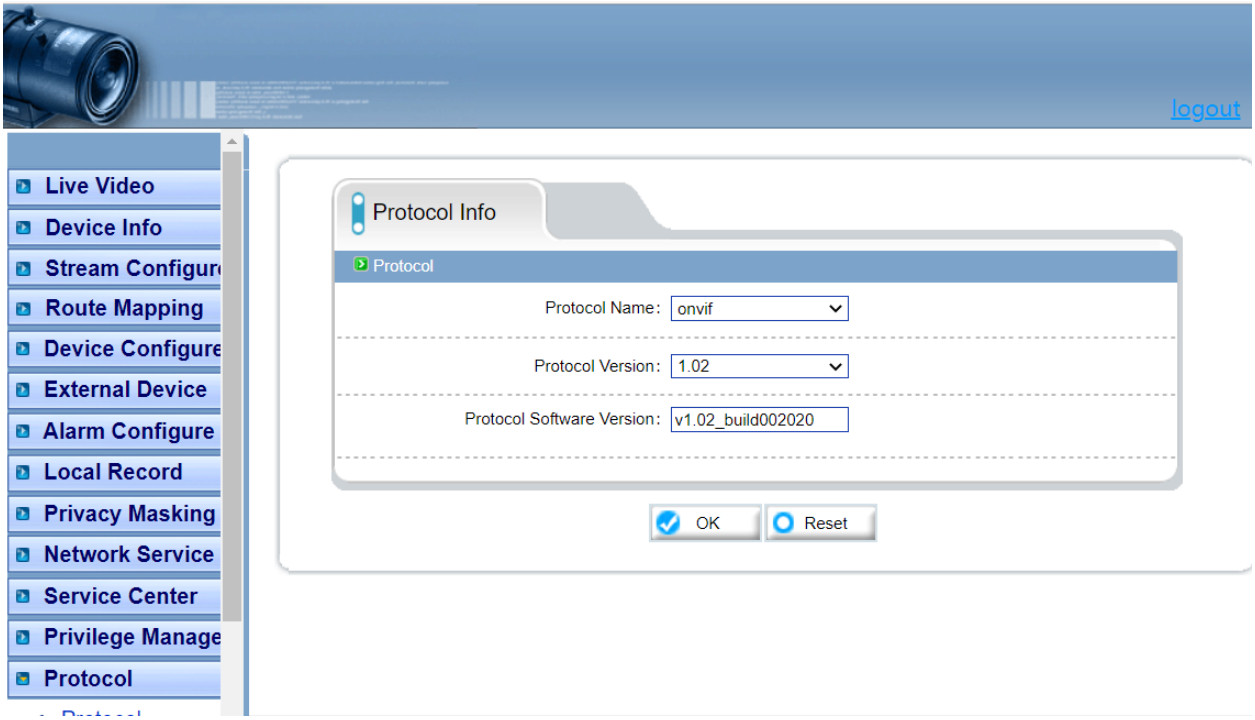
Kod:

```
import cv2 as cv
stream = cv.VideoCapture ('rtsp://192.168.0.200:554/snl/live/1/1/Ux/sido=-Ux/sido=')
while True:
   check, frame = stream.read()
   cv.imshow('output', frame)
   key = cv.waitKey(0)
   if key == ord('x'):
        break
```

Slika 50 - Slika sa kamere unutar Python-a

6.2.1.2. Uspostavljanje upravljanja kamere putem *Python-a*

Nakon što se uspije riješiti problem dobivanja slike sa kamere, potrebno je uspostaviti drukčiju vezu sa kamerom, vezu preko koje će se kameri slati naredbe, tj. funkcije koje će kamera izvršavati. Kao što je već prije rečeno, to radimo pomoću ONVIF standarda. Da kamera radi putem ONVIF standarda moguće je vidjeti u web sučelju kamere.



Slika 51 - Prikaz upravljačkog protokola kamere u web sučelju kamere

Kao što je ranije rečeno, upravljački ONVIF modul unutar *Python-a* koji je potrebno skinuti se zove *onvif-py3*. Nakon njegove instalacije je moguće koristiti njegove funkcionalnosti. Za uspostavljanje veze sa kamerom je potrebno znati više parametara radi identifikacije i sigurnosti. Ti su parametri IP adresa kamere, *port*, korisničko ime i zaporka pomoću koje se ulazi u korisničko web sučelje kamere. Osim navedenih podataka, potrebno je definirati servise na kameri koji će se koristiti kako bi se prilikom pozivanja određenih funkcija znali točno svi parametri potrebni za izvršavanje funkcije. Potrebno je u programskom kodu definirati objekt, odnosno servis *media* kako bi se identificirao profil na kameri. Moguće je kreirati više profila na kameri gdje svaki profil posjeduje svoje postavke te je bez određivanja profila sa pripadajućim postavkama nemoguće izvršavati funkcije na kameri. Kada se kreira objekt *media,* tada je potrebno iz servisa *media* odabrati profil preko kojega će se upravljati kamerom. Nakon definiranja profila, moramo znati kakve funkcije želimo izvršavati na kameri. S obzirom da se u ovom zadatku želi pokretati kamera u svim smjerovima, potrebno je kreirati objekt PTZ. To je servis na kameri koji je zadužen za funkcije pokretanja. Unutar tog

servisa definiramo funkcije pokretanja kamere u željenom smjeru. Funkcija koju smo iskoristili za pokretanje kamere zove se *Continuous move.* To znači da će se kamera kretati u definiranom smjeru dok joj ne pošaljemo zahtjev da se prestane gibati. Pomoću te funkcije postižemo gibanje kamere sve dok se gibajući objekt ne nađe unutar centra slike kamere te se tada šalje naredba da se kamera prestane gibati. Dakle, nakon definiranja servisa *media*, odabiranja profila iz servisa *media*, nakon definiranja servisa PTZ, potrebno je kreirati objekt u koji ćemo spremiti informacije o načinu pokretanja kamere. U proizvoljni objekt spremamo informacije o načinu gibanja kamere, odnosno funkciju *Continuous move* (u ovom slučaju), prethodno spomenuti profil sa kojeg upravljamo kamerom te brzine gibanja kamere u smjeru x i y. Za funkciju *Stop* koja zaustavlja sve radnje kamere, potrebno je definirati unutar novog objekta profil sa kojeg se upravlja kamerom te iz servisa PTZ odabrati funkciju *Stop.* Na ovaj način možemo odrediti gibanja kamere u svim smjerovima pomoću definiranja brzina gibanja po smjerovima x i y te zaustavljanje kamere pomoću funkcije *Stop*.

Ispod će biti prikazan kod koji omogućuje pokretanje kamere pritiscima na tipke sa strelicama na tipkovnici. Program je napravljen tako da se kamera giba sve dok je tipka pritisnuta, a dok se tipka otpusti program šalje kameri signal za prestanak gibanja. Ovo je bilo napravljeno u ranijim fazama izrade rada te je upravljanje tipkama nepotrebno za potrebe rada jer će konačni program sam određivati smjerove i vremena kretanja kamere. Iz tog razloga ovaj programski kod i način funkcioniranja neće biti detaljno opisani, ali zornosti radi te zbog načina definiranja parametara kamere, kod će biti prikazan u cijelosti.

Kod:

```
from onvif import ONVIFCamera
import keyboard
import time


cam = ONVIFCamera('192.168.0.200', 80, 'admin', 'admin')


media = cam.create_media_service()
ptz = cam.create_ptz_service()
profile = media.GetProfiles()[0]
block = False
exit_prog = True
keyValues = ['up', 'down', 'left', 'right', 'z']
keyValuesNum = [72, 75, 80, 77]
```

```
def MoveLeft():
    contMove = ptz.create_type('ContinuousMove')
    contMove.ProfileToken = profile.Name
    contMove.Velocity.PanTilt._x = -0.4
    contMove.Velocity.PanTilt._y = 0.0
    ptz.ContinuousMove(contMove)
    print('Moving left')

def MoveRight():
    contMove = ptz.create_type('ContinuousMove')
    contMove.ProfileToken = profile.Name
    contMove.Velocity.PanTilt._x = 0.4
    contMove.Velocity.PanTilt._y = 0.0
    ptz.ContinuousMove(contMove)
    print('Moving right')

def MoveUp():
    contMove = ptz.create_type('ContinuousMove')
    contMove.ProfileToken = profile.Name
    contMove.Velocity.PanTilt._x = 0.0
    contMove.Velocity.PanTilt._y = 0.4
    ptz.ContinuousMove(contMove)
    print('Moving up')

def MoveDown():
    contMove = ptz.create_type('ContinuousMove')
    contMove.ProfileToken = profile.Name
    contMove.Velocity.PanTilt._x = 0.0
    contMove.Velocity.PanTilt._y = -0.4
    ptz.ContinuousMove(contMove)
    print('Moving down')

def Stop():
    stop = ptz.create_type('Stop')
    stop.ProfileToken = profile.Name
```

```
    ptz.Stop(stop)

def ZoomIn():
    zIn = ptz.create_type('ContinuousMove')
    zIn.ProfileToken = profile.Name
    zIn.Velocity.Zoom._x = 0.3
    ptz.ContinuousMove(zIn)
    print('Zooming in')

def ZoomOut():
    zOut = ptz.create_type('ContinuousMove')
    zOut.ProfileToken = profile.Name
    zOut.Velocity.Zoom._x = -0.3
    ptz.ContinuousMove(zOut)
    print('Zooming out')

def stopCameraMove(event):
    global block
    print('Stop')
    Stop()
    if event.name in keyValues:
        block = False

def startCameraMove(event):
    global block, exit_prog
    if not block:
        if event.name == "up":
            block = True
            MoveUp()

        if event.name == 'left':
            block = True
            MoveLeft()
```

```
        if event.name == 'right':
            block = True
            MoveRight()

        if event.name == 'down':
            block = True
            MoveDown()

        if event.name == '+':
            block = True
            ZoomIn()

        if event.name == '-':
            block = True
            ZoomOut()

        if event.name == 'e':
            print('Exit program')
            exit_prog = False

def handle_event(event):
    if event.event_type == 'down':
        startCameraMove(event)
    if event.event_type == 'up':
        stopCameraMove(event)

keyboard.hook(handle_event)

while exit_prog:
    time.sleep(0.2)
```

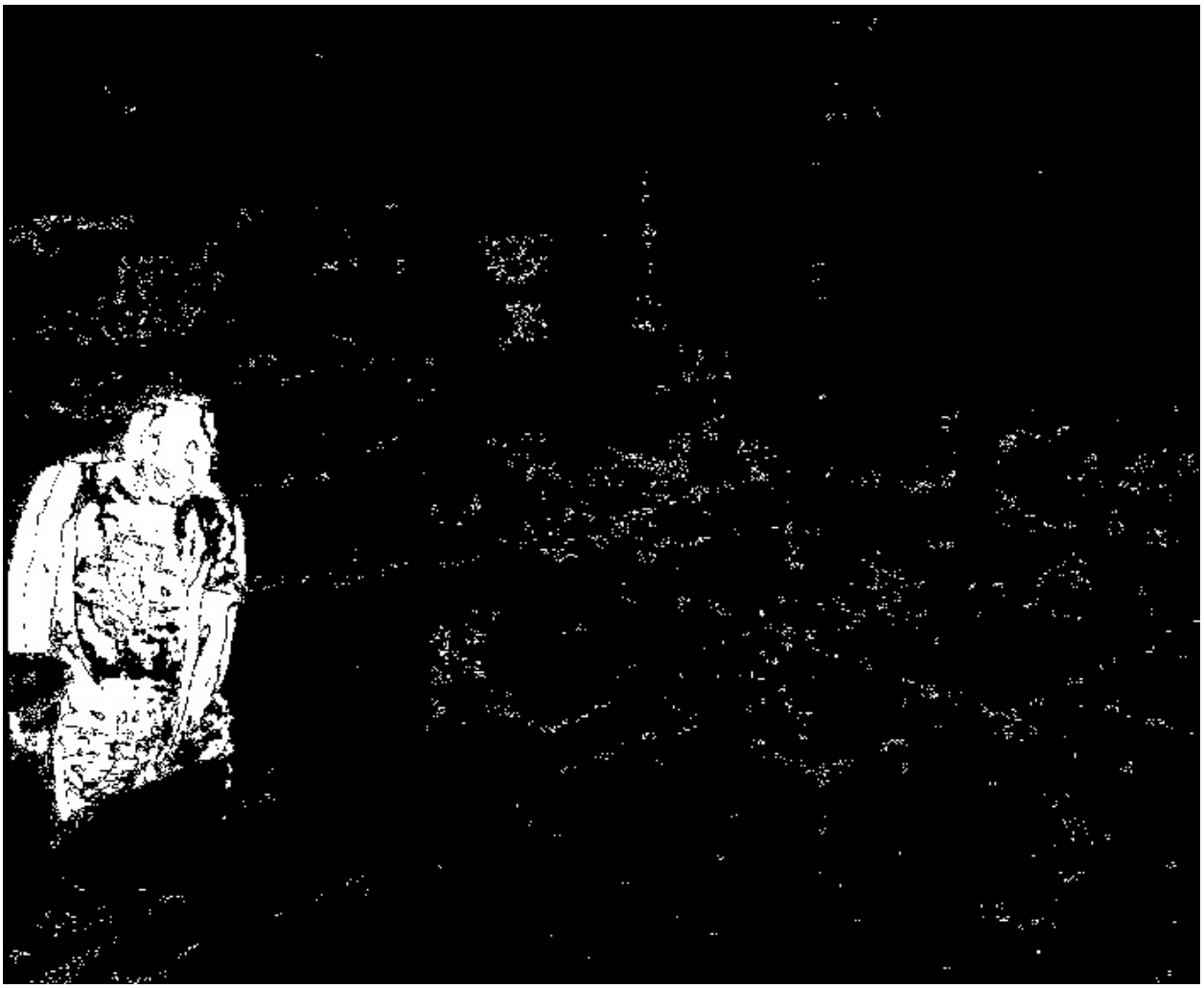
6.2.2. Detekcija kretanja na slici

Nakon što se riješi upravljanje kamerom i dobije slika sa kamere, može se prijeći na idući korak. Idući korak je detektirati gibanje na slici dok je kamera statična. Dok je kamera statična slika se ne mijenja. Ukoliko dolazi do promjene na slici znamo da se nešto pomaknulo. Neki objekt je obavio gibanje ili se giba i dalje. Ako se objekt giba, u početnom trenutku gibanja t_0 objekt će biti na poziciji x_0. U trenutku t_1 objekt će se nalaziti na poziciji x_1. To znači da će se u istom kadru, dok je kamera statična, objekt pomaknuti za određeni iznos x_1-x_0. U tom slučaju, s obzirom da je kamera statična, okolina na slici će uvijek biti ista. Dijelovi slike u trenutku t_0 i t_1 će se razlikovati na onim mjestima na kojima se objekt micao. To znači da možemo oduzeti jednu sliku od druge, odnosno vrijednosti piksela slike t_0 i t_1, a krajnji rezultat će biti matrica u kojoj će se nalaziti nule za dijelove slike koji se nisu mijenjali, a tamo gdje se objekt pomaknuo vrijednost piksela će biti različita od nule. Kako bi rad sa slikom bio jednostavniji, slike koje dobivamo sa kamere pretvaramo u *grayscale* sustav boja. Time osiguravamo opis slike jednom matricom sa pikselima vrijednosti između 0 i 255. Rezultantna slika, koju dobijemo oduzimanjem slike t_0 i t_1, će biti crna na svim područjima gdje se ništa nije promijenilo, a tamo gdje je došlo do promjene na slici biti će vidljivo. Biti će vidljivi dijelovi slike t_0 i t_1 čije se vrijednosti piksela ne podudaraju jer će se oduzimanjem pripadajućih piksela dobiti razlika različita od 0. Ako je razlika piksela veća, znači da će biti veći intenzitet piksela na krajnjoj slici te će područja intenzivnijih promjena na slici biti jače prikazana. Ako uzmemo mali vremenski period Δt, u kojem dohvatimo sliku t_0 i t_1 sa kamere, pomak koji će objekt napraviti će biti relativno malenog iznosa te će stoga rezultat oduzimanja te dvije slike biti dovoljno precizan obris objekta koji se giba. Dobivena regija kao rezultat oduzimanja će prikazivati obrise objekta relativno precizno. U slučaju velikog vremenskog perioda Δt, pomak objekta bi bio relativno velik. U tom slučaju bi se moglo dogoditi da se na rezultantnoj slici pojave dvije potpuno odvojene regije koje označavaju pozicije objekta na slici t_0 i t_1. Ako bi na rezultantnoj slici bile dvije potpuno odvojene regije, ne bi se moglo sa lakoćom odrediti sadašnja pozicija objekta bez daljne obrade slika t_0 i t_1. Iz tog razloga uzimamo mali vremenski period Δt u kojem će pomak objekta biti malen kako bi dobivena regija na rezultatnoj slici dovoljno precizno pokazala položaj objekta. S obzirom da bi uzeli mali vremenski period Δt, pomak objekta na slici bi također bio mali te bi na rezultantnoj slici dobili dva obrisa objekta koji su zamaknuti za mali iznos gibanja objekta kroz period Δt. Time dobijemo preciznu lokaciju gibajućeg objekta na slici te njegov oblik. Za uočavanje razlika na dvije slike postoji funkcija unutar *OpenCV* modula *absdiff*. Parametri te funkcije su dvije slike za koje tražimo područja razlikovanja. Prikaz rezultata funkcije *absdiff* je prikazan na Slika 52.

Slika 52 - Rezultat funkcije absdiff

Kao što vidimo sa slike, obris čovjeka je lako prepoznati. Oblici su vidljivi, vide se 2 konture malo zamaknute te je stoga lagano odrediti njegov položaj na slici nakon gibanja. Nakon toga primjenjujemo funkciju *threshold.* Kako rezultantna slika na područjima gdje se nalaze pomaci objekta sadrži vrijednosti piksela različite od nule, koristimo navedenu funkciju kako bi sve piksele čija je vrijednost različita od nule pretvorili u jednu cjelinu. Ta funkcija će sliku pretvoriti u binarnu. Pikseli rezultantne slike čija je vrijednost 0 će ostati 0, a ostala područja slike gdje pikseli imaju drugu vrijednost će postati 1. Time ćemo dobiti jednu smislenu cjelinu, oblik objekta na slici sa kojim je lakše dalje raditi. Ako uzmemo u obzir na neke manje promjene u okolini, npr. promjena svjetlosti na zidu ukoliko se zavjesa pomakne, loša rezolucija kamere ili nešto slično, problem se djelomično rješava postavljanjem intervala za funkciju *threshold*. Postavljaju se intervali tako da pikseli čija je vrijednost 0-5 dobiju vrijednost 0, a pikseli sa ostalim vrijednostima poprime vrijednost 1. Time uklanjamo manje šumove koji mogu nastati iz nekih drugih razloga. Interval 0-5 je moguće i povećati kako bi se još više uklonio šum sa slike, ali tada gubimo dosta informacija sa slike te je stoga odabran takav interval. Odabrani vremenski interval Δt je 0.2 s, odnosno uzimamo početnu i petu sliku. S obzirom da kamera radi sa 25 FPS-a, a program računa

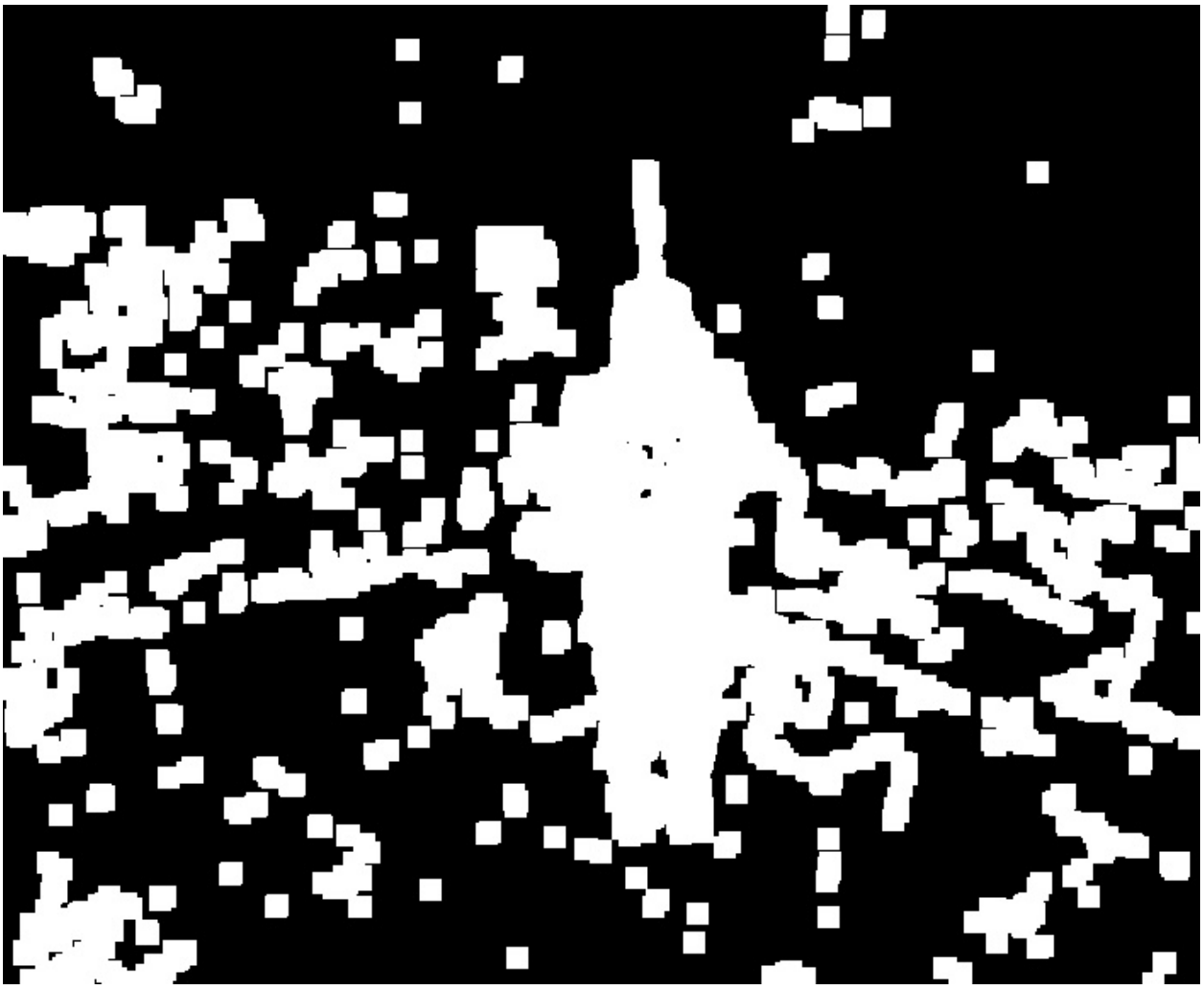
razlike svake prve i pete slike, program u jednoj sekundi može 5 puta detektirati gibanje. Rezultat nakon primjene funkcije *threshold* je dan na



Slika 53 - Rezultat funkcije threshold

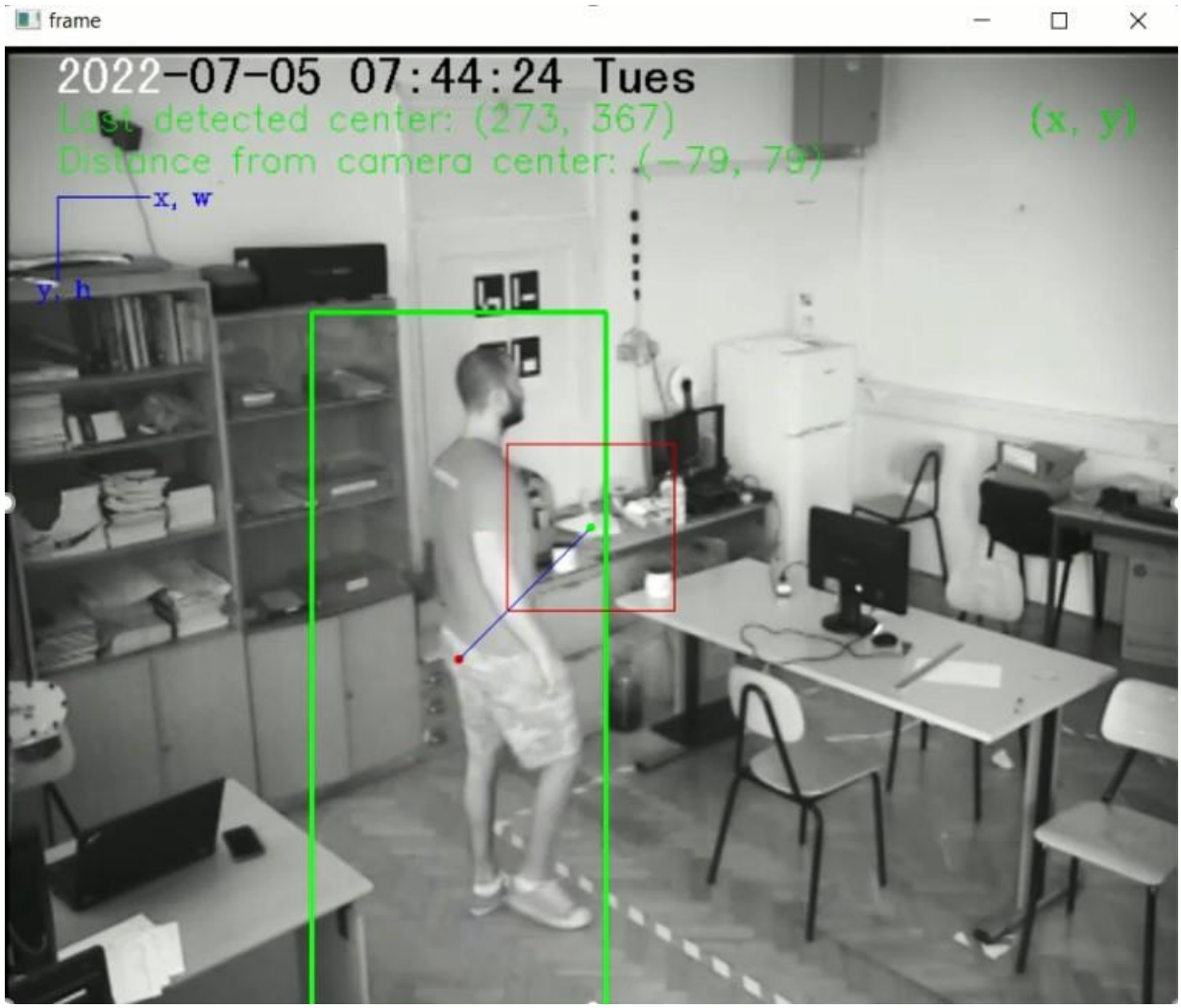
Kao što vidimo sa slike, rezultat funkcije nije jedna spojena cjelina, već odvojeni dijelovi objekta te dosta okolnog šuma koji nastaju zbog niske rezolucije kamere ili razlika u osvijetljenju dviju slika. Vidimo da je čovjek prikazan pomoću više većih i odvojenih regija. Nekada su te regije više zbijene, a nekada manje. Iz tog razloga želimo površinu čovjeka dobiti u što homogenijem obliku, u najboljem slučaju kao jednu veliku površinu obrisa čovjeka. Osim toga, vidimo na slici preostali šum čiji su intenziteti piksela veći od vrijednosti 5. Iako se to nije moglo raspoznati na Slika 52 jer su pikseli vrijednosti od npr. 5 do 20, ljudskom oku veoma slični crnoj boji te ih je teško raspoznati. Nakon primjene funkcije *threshold*, kada svi pikseli poprime vrijednost 0 ili 1, taj šum dolazi do izražaja. Koristit će se funkcija *dilation* kako bi se dobivene regije na slici 53 povećale. Ova funkcija radi tako da se određene konture podebljavjau za željeni iznos. Tako će se regije koje opisuju čovjeka, za dovoljno veliku dodanu debljinu konture, spojiti u jendu cjelinu. Cilj ovoga je

bijelu površinu čovjeka spojiti u jednu veliku regiju kako bi se kasnije mogao filtrirati šum pomoću parametara veličine površine regije. Program će u konačnici uzimati u obzir samo one konture čija je površina veća od 4000 px. Time ćemo ukloniti šum sa slike, a preostati će samo čovjek i eventualno još neke promjene koje će površinom biti manje od čovjeka, a veće od 4000 px. Stoga će program uvijek odabrat najveću gibajuću površinu. Prikaz rezultata funkcije *dilation*, koja je primjenjena na Slika 53., je dan na Slika 54.



Slika 54 - Rezultat primjene funkcije dilation

Kao što vidimo sa slike, konture su podebljane. Područje čovjeka se spojilo u jednu cjelinu, ali su se povećala i područja šuma. Sada je jednostavno izdvojiti najveću površinu. Kada se dobije jedna homogena površina na slici lako je odrediti njen centar, površinu, visinu i širinu. Kako bi grafički prikazali dobiveno rješenje, oko dobivene površine ćemo opisati pravokutnik. Za to koristimo ranije navedene funkcije crtanja oblika. Potrebno je znati visinu, širinu i centar površine oko koje želimo opisati pravokutnik kako bi izračunali točke pravokutnika. To radimo pomoću funkcije *rectangle* unutar modula *OpenCV*. Krajnji rezultat je prikazan na Slika 55. Programski kod za ovaj problem je dan ispod slike.



Slika 55 - Detekcija gibanja

Kod:

```
import cv2

import numpy as np

import os


frame_count = 0

switch = 0

img_before = None

img_now = None

center_roi_x = 50
```

```
center_roi_y = 50
moving_object_center = None
biggest_moving_object_center = None
distance_from_frame_center = None

try:
    print('trying to connect to the camera')
    cam = cv2.VideoCapture('rtsp://192.168.0.200:554/snl/live/1/1/Ux/sido=-Ux/sido=')
    print('camera connected')

    while True:

        check, frame = cam.read()
        check2, frame2 = cam.read()
        height, width, layers = frame.shape
        frame_center = (int(width / 2), int(height / 2))
        img_diff = None

        # blurring img for reducing noise on the image
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        gray = cv2.GaussianBlur(gray, (21, 21), 0)

        # catching frame fps=1
        if switch == 1:
            img_before = gray

        # catching frame fps=5
        if switch == 5:
```

```
        img_now = gray
        switch = 0

        # getting motion in form of pixels, changing it to binary
        # getting them into a uniform area and making contours

        img_diff = cv2.absdiff(img_now, img_before)
        motion_threshold = cv2.threshold(img_diff, 5, 255, cv2.THRESH_BINARY)[1]
        threshold_dilated = cv2.dilate(motion_threshold, np.ones((5,5), np.uint8), iterations = 3)
        contours, hierarchy = cv2.findContours(threshold_dilated, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

        # filtering out smaller ones, getting the biggest
        cv2.drawContours(img_diff, contours, -1, (255, 255, 255), 2)
        biggestArea = 0

        for cnt in contours:
            cntArea = cv2.contourArea(cnt)

            if cntArea > 4000:
                x, y, w, h = cv2.boundingRect(cnt)

                if h > w:
                    moving_object_center = (int(x + w / 2), int(y + h / 2))
                    cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
                    cv2.circle(frame, moving_object_center, 1, (0, 0, 255), 2)
                    cv2.line(frame, frame_center, moving_object_center, (255, 0, 0), 1)

                    if cntArea > biggestArea:
```

```
                biggestArea = cntArea

                biggestContour = cnt

                x, y, w, h = cv2.boundingRect(cnt)

                biggest_moving_object_center = (int(x + w / 2), int(y + h / 2))

                distance_from_frame_center = ((biggest_moving_object_center[0] - frame_center[0]),
(biggest_moving_object_center[1] - frame_center[1]))


    switch = switch + 1


    cv2.putText(frame, f'Last detected center: {biggest_moving_object_center}', (33, 50),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 1)

    cv2.putText(frame, f'Distance from camera center: {distance_from_frame_center}', (33, 75),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 1)

    cv2.circle(frame, (int(width/2), int(height/2)), 1, (0, 255, 0), 2)

    cv2.rectangle(frame, (frame_center[0]+center_roi_x, frame_center[1]+center_roi_y),
(frame_center[0]-center_roi_x, frame_center[1]-center_roi_y), (0, 0, 255), 1)

    cv2.line(frame, (33, 90), (33, 140), (255, 0, 0), 1)

    cv2.line(frame, (33, 90), (88, 90), (255, 0, 0), 1)

    cv2.putText(frame, 'x, w', (90, 95), cv2.FONT_HERSHEY_COMPLEX, 0.5, (255, 0, 0), 1)

    cv2.putText(frame, 'y, h', (20, 150), cv2.FONT_HERSHEY_COMPLEX, 0.5, (255, 0, 0), 1)

    cv2.putText(frame, '(x, y)', (width-90, 50), cv2.FONT_HERSHEY_COMPLEX, 0.7, (0, 255, 0), 1)


    cv2.imshow('framediff', img_diff)

    cv2.imshow('motion', threshold_dilated)

    cv2.imshow('webcam', gray)

    cv2.imshow('frame', frame)


    key = cv2.waitKey(1)

    path = "C:/Users/CAD/PycharmProjects/pythonProject2/screenshots"

    if key == ord('x'):
```

```
            break


except:
    print('Can\'t connect to the camera.')


cam.release()
cv2.destroyAllWindows()
```

6.2.3. Referentne točke u prostoru za određivanje udaljenosti objekta

Nakon detekcije gibanja objekta na slici potrebno je definirati prostor u kojemu se predmet nalazi kako bi program bio u stanju računati udaljenosti objekta u odnosu na točku u prostoru i u odnosu na središte slike. Tako smo spojili apsolutni i lokalni koordinatni sustav.

Za referentne točke u prostoru je bilo potrebno definirati neke objekte ili površine koji su statični i specifični. U tu svrhu su odabrani predmeti i napravljene oznake kako bi s lakoćom mogli odrediti njihov položaj na slici. Po cijeloj širini prostorije je bilo potrebno postaviti oznake i odabrati predmete tako da u svakom trenutku kamera u nekom položaju uspije identificirati minimalno jedan referentni objekt. Odabir referentnih objekata je prikazan na slikama ispod:

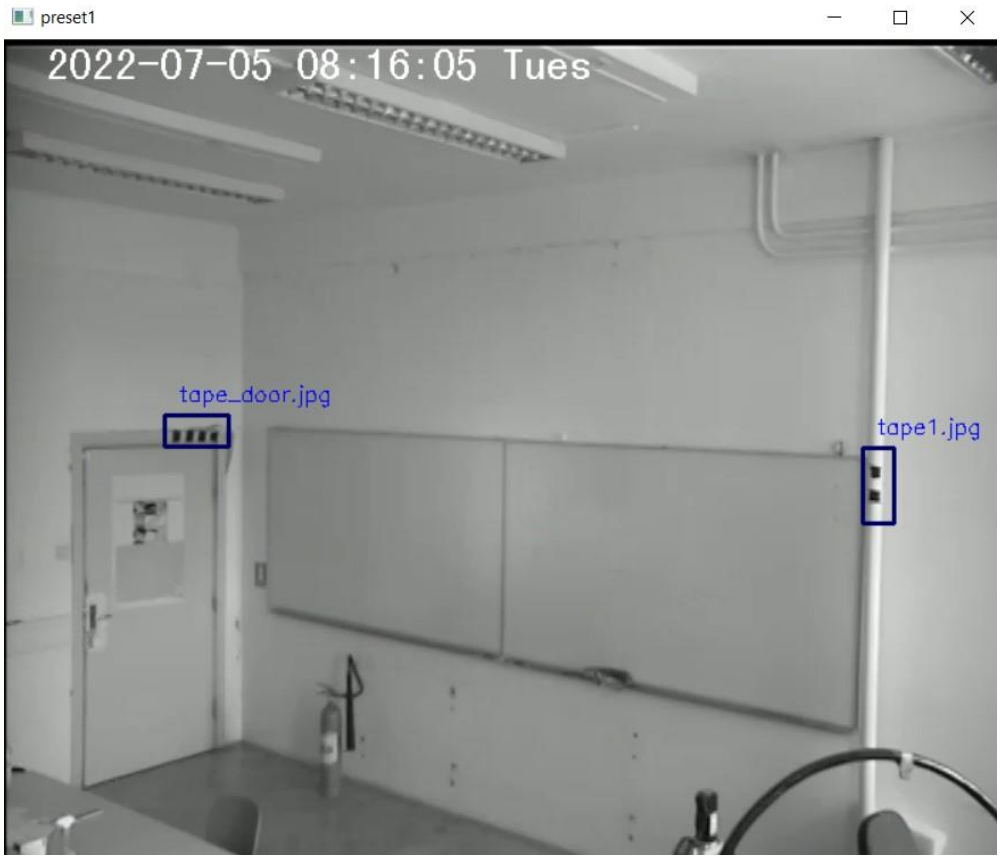
Slika 56 - Referentni objekti u prostoru

Redom kako su prikazani objektima je dan naziv: vr_sensor, tape1, tape_window, tape_tv, tape_door, hanger, aruco2, aruco, vr_sensor2. Objekti se nalazi po cijeloj širini prostorije kako bi pokrili svaki horizontalni kut kamere u kojem se čovjek može gibati po učionici, a da ga kamera uspije pratiti. Slike navedenih objekata su izrezane sa slika sa kamere. Funkcija kojom se nalaze dani objekti se zove *match template*, a o tome je ranije bila riječ. Program na svakom *frame-u*, tj. slici nalazi objekte koji su u kadru, a ukoliko program detektira gibanje tada računa udaljenost objekta od nađene referente točke sa slike.

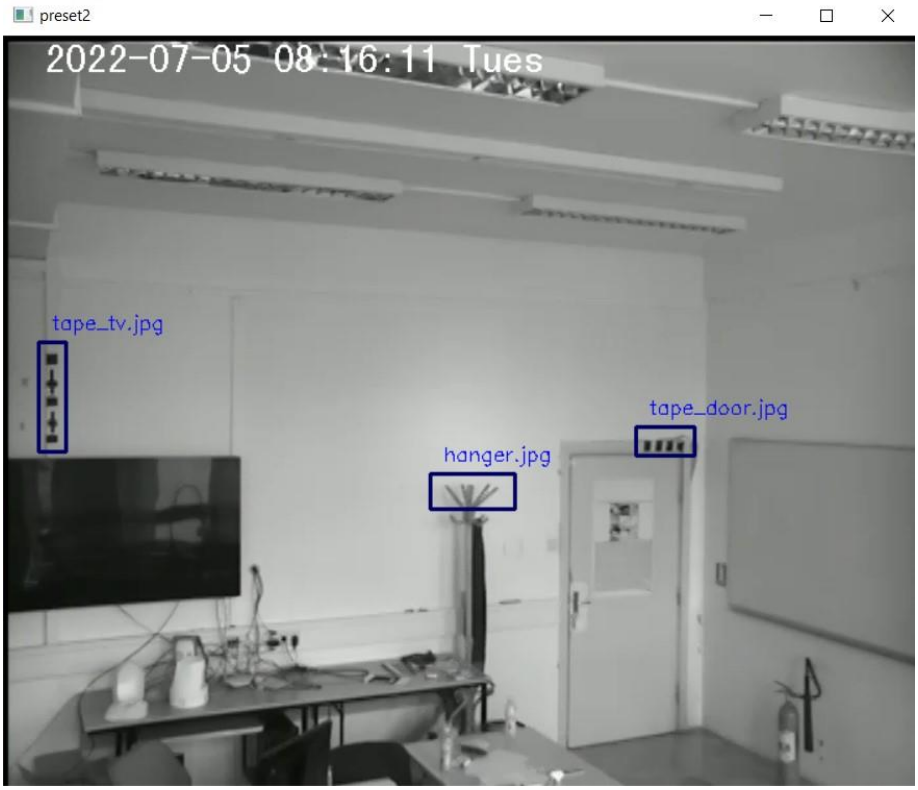
S obzirom da funkcija *match template* radi po principu vjerojatnosti podudaranja, potrebno je definirati granične vrijednosti. U tu svrhu je bilo potrebno ispitati minimalne vrijednosti identifikacije pojedinih objekata. Pojedini objekti su bili postavljani u razne dijelove kadra slike te su bile očitane vrijednosti. Na taj način su se uzeli podaci koji služe kao granične vrijednosti. Unatoč tome, program nekada nađe krivi dio slike za veću vjerojatnost podudaranja od minimalne. Taj se problem riješio programskim kodom kojeg je potrebno izvršiti prije početka detekcije gibanja i naknadnog praćenja gibanja kamerom.

Programski kod koji se izvršava prije glavnog programskog koda služi za 'mapiranje' prostora u kojem kamera vrši detekciju referentnih objekata po prostoru. Na kameri je bilo određeno nekoliko *preset* pozicija. *Preset* pozicije su unaprijed određeni i zapamćeni smjerovi kamere koje je moguće pohraniti u sustavu kamere. Na taj način, nakon što se takva jedna pozicija zapamti, moguće ju je naknadno pozvati da se kamera postavi u takvu poziciju. Programski kod uzima slike izrezanih objekata koje su prethodno pohranjene u mapu na računalu te ih pronalazi na različitim *preset* pozicijama. Kod je napravljen tako da

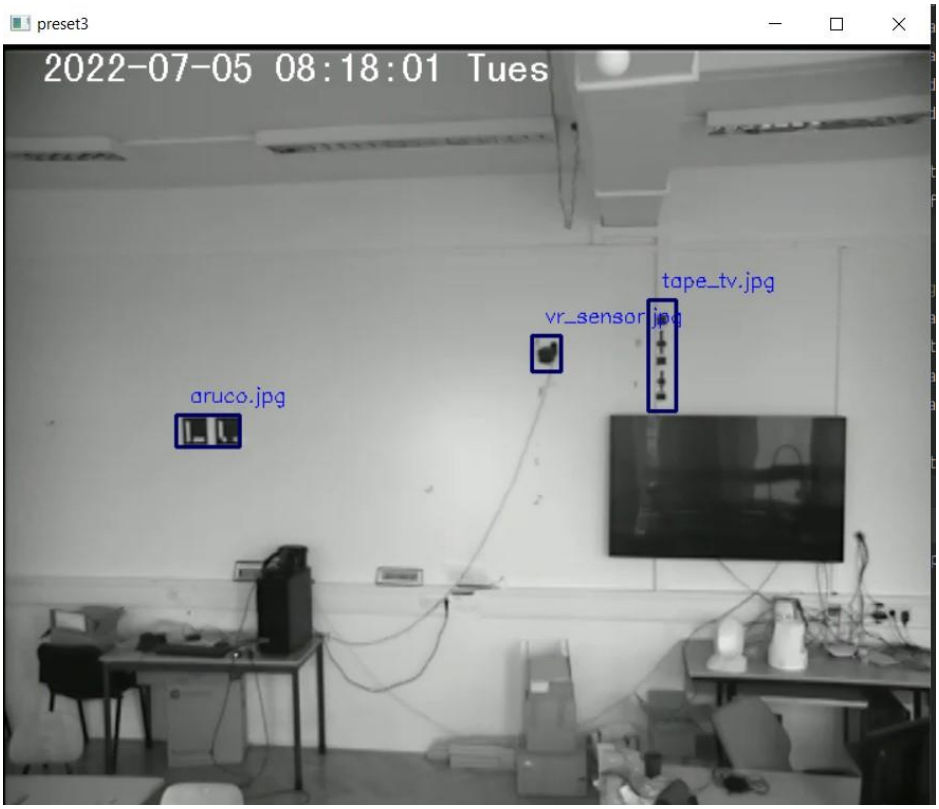
se kamera postavlja u početni položaj, krajnji desni položaj prostorije, te identificira referentne objekte koji su u kadru. Unutar programa se nalazi lista objekata koje je potrebno pronaći, a kada ih se pronađe, nađene objekte se upisuje u novu listu. Na taj način se reducira prostor za pogrešku, odnosno da se određeni predmet nađe na više pozicija. Kada program detektira više referentnih predmeta unutar jedne pozicije tada se računaju međusobne udaljenosti tih objekata te se zapisuju njihovi relativni odnosi. Relativni odnosi se zapisuju za svaku referentnu točku u obliku lijevi susjed i desni susjed te njihovi međusobni iznosi udaljenosti po x i y osi. Osim toga, program u posebnu listu upisuje redoslijed referentnih točki u prostoru, onako kako se pojavljuju, s lijeva na desno. Kada se izvrši takav programski kod, posjedujemo sve potrebne podatke za lakše identificiranje i praćenje gibajućih objekata na slici. Zapisani su podaci o redoslijedu referentnih objekata, podaci o međusobnim udaljenostima referentnih točki te njihovi relativni odnosi. Ti će se podaci koristiti u glavnom programu za provjeravanje, uklanjanje grešaka i bolje upravljanje. Slike *preset* pozicija sa nađenim referentnim objektima su dane ispod kao i programski kod.



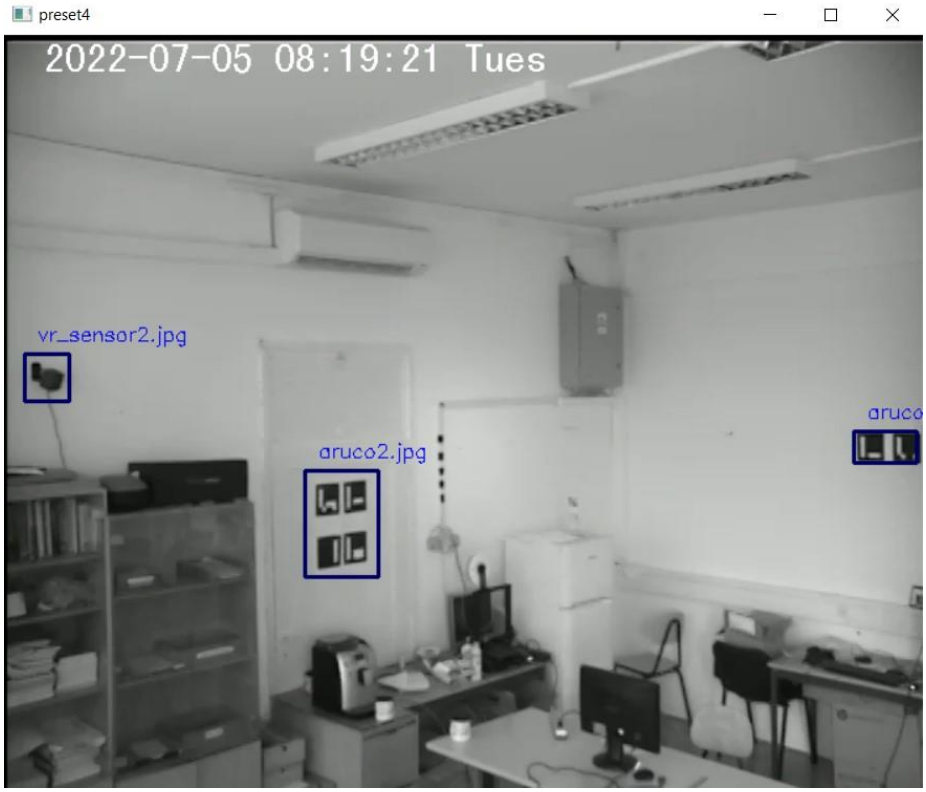
Slika 57 - Preset 1



Slika 58 - Preset 2



Slika 59 - Preset 3



Slika 60 - Preset 4

Slika 61 - Preset 5

Kod:

```
 import os
import time
import cv2 as cv
from onvif import ONVIFCamera

# list of the objects that will be found
cwd = os.getcwd()
objects_dir = "/objects"
objects = os.listdir(cwd + objects_dir)
objects_left = objects[:]
objects_data = {}

#match template minimum result for each referent object
results_min = { "tape1.jpg": 0.82,
                "tape_door.jpg": 0.80,
                "hanger.jpg": 0.790,
                "tape_tv.jpg": 0.81,
                "vr_sensor.jpg": 0.80,
                "aruco.jpg": 0.75,
                "electric_box.jpg": 0.82,
                "vr_sensor2.jpg": 0.82,
                "aruco2.jpg": 0.9773,
                "tape_window.jpg": 0.79}

# connecting to camera – ONVIF controls
cam = ONVIFCamera('192.168.0.200', 80, 'admin', 'admin')
media = cam.create_media_service()
ptz = cam.create_ptz_service()
profile = media.GetProfiles()[0]

# getting saved preset positions
get_presets = ptz.create_type('GetPresets')
```

```
get_presets.ProfileToken = profile.Name
presets = ptz.GetPresets(get_presets)

# controls for putting camera in each preset position for finding referent objects
num_of_presets = len(presets)
go_to_preset = ptz.create_type('GotoPreset')
go_to_preset.ProfileToken = profile.Name

counter = 0
object_order = []
object_temp_order = []
object_coordinates_x = []
object_coordinates_y = []
to_find_in_next = []
object_relations = {}

for i in presets[:-1]:
    object_temp_order = []
    objs_dict = {}
    counter += 1
    go_to_preset.PresetToken = i.Name
    ptz.GotoPreset(go_to_preset)
    time.sleep(3)
    if i == presets[0]:
        time.sleep(1)

    # getting camera video - OpenCV
    video = cv.VideoCapture('rtsp://192.168.0.200:554/snl/live/1/1/Ux/sido=-Ux/sido=')
    check, frame = video.read()
    gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
    obj_found = []
    obj_found_x = []
    obj_found_y = []
```

```
# search for every object in the frame
for obj in objects_left:

    obj_img = cv.imread(cwd + objects_dir + "/" + obj)
    height, width, layers = obj_img.shape
    obj_img_gray = cv.cvtColor(obj_img, cv.COLOR_BGR2GRAY)

    res = cv.matchTemplate(obj_img_gray, gray, cv.TM_CCOEFF_NORMED)
    min_val, max_val, min_loc, max_loc = cv.minMaxLoc(res)
    obj_val = results[obj]

    if abs(max_val - obj_val) < 0.08:
        x = max_loc[0]
        y = max_loc[1]
        obj_found.append(obj)
        obj_found_x.append(x)
        obj_found_y.append(y)
        objs_dict.update({x: [obj, y]})
        objects_data.update({obj: [x, y, height, width]})
        cv.rectangle(frame, (x, y), (x + width, y + height), 100, 2)
        cv.putText(frame, f'{obj}', (x + 10, y - 10), cv.FONT_HERSHEY_SIMPLEX, 0.5, 255, 1)

# find the most left object from the last frame in the next frame
if len(to_find_in_next) != 0:
    for obj in to_find_in_next:
        obj_img = cv.imread(cwd + objects_dir + "/" + obj)
        height, width, layers = obj_img.shape
        obj_img_gray = cv.cvtColor(obj_img, cv.COLOR_BGR2GRAY)

        res = cv.matchTemplate(obj_img_gray, gray, cv.TM_CCOEFF_NORMED)
        min_val, max_val, min_loc, max_loc = cv.minMaxLoc(res)
        obj_val = results[obj]
```

```
            x = max_loc[0]
            y = max_loc[1]

            objs_dict.update({x: [obj, y]})
            obj_dict_update_list = objects_data[obj]
            obj_dict_update_list.append(x)
            obj_dict_update_list.append(y)
            objects_data.update({obj: obj_dict_update_list})
            cv.rectangle(frame, (x, y), (x + width, y + height), 100, 2)
            cv.putText(frame, f'{obj}', (x + 10, y - 10), cv.FONT_HERSHEY_SIMPLEX, 0.5, 255, 1)
            to_find_in_next.remove(obj)

    # delete found objects from the list
    for obj in obj_found:
        objects_left.remove(obj)

    # order found objects in the right order
    objs_dict_sorted = dict(sorted(objs_dict.items(), key=lambda item: item[0]))
    objs_dict_inverse = {}

    # ordering found objects from the frame
    for j in objs_dict_sorted:
        obj_name_list = objs_dict_sorted.get(j)
        obj_name = obj_name_list[0]
        if obj_name not in object_order:
            object_temp_order.append(obj_name)

    # calculating relations between neighbouring referent objects
    # save referent object order in the global order
    object_temp_order.extend(object_order)
    object_order = object_temp_order[:]
```

```
    # object with the minimum value will be found again in the next frame
    minimum = min(obj_found_x)
    index = obj_found_x.index(minimum)
    to_find_in_next.append(obj_found[index])

    # show
    cv.imshow('preset' + str(counter), frame)
    cv.waitKey(1)

for i in range(len(object_order) - 1, -1, -1):

    obj = object_order[i]
    obj_data = objects_data[obj]

    if i == (len(object_order) - 1):
        obj_next = object_order[i - 1]
        obj_next_data = objects_data[obj_next]
        object_relations.update({obj: {'left': obj_next, 'right': None,
                                'dist_left': [-obj_data[0] + obj_next_data[0], -obj_data[1] + obj_next_data[1]],
                                'dist_right': None, 'height': obj_data[2], 'width': obj_data[3],
                                'height_left': obj_next_data[2], 'width_left': obj_next_data[3],
                                'height_right': None, 'width_right': None}})

    elif i == 0:
        obj_before = object_order[i + 1]
        obj_before_data = objects_data[obj_before]
        object_relations.update({obj: {'left': None, 'right': obj_before, 'dist_left': None,
                                'dist_right': [-(obj_data[0] - obj_before_data[4]),
                                        -(obj_data[1] - obj_before_data[5])], 'height': obj_data[2],
                                'width': obj_data[3], 'height_left': None, 'width_left': None,
                                'height_right': obj_before_data[2], 'width_right': obj_before_data[3]}})

    else:
```

```
obj_next = object_order[i - 1]
obj_next_data = objects_data[obj_next]
obj_before = object_order[i + 1]
obj_before_data = objects_data[obj_before]

if len(obj_data) > 5:
   object_relations.update({obj: {'left': obj_next, 'right': obj_before,
                        'dist_left': [-(obj_data[4] - obj_next_data[0]),
                                -(obj_data[5] - obj_next_data[1])],
                        'dist_right': [-obj_data[0] + obj_before_data[0],
                                 -obj_data[1] + obj_before_data[1]], 'height': obj_data[2],
                        'width': obj_data[3], 'height_left': obj_next_data[2],
                        'width_left': obj_next_data[3], 'height_right': obj_before_data[2],
                        'width_right': obj_before_data[3]}})

elif len(obj_before_data) > 5:
   object_relations.update({obj: {'left': obj_next, 'right': obj_before,
                        'dist_left': [-(obj_data[0] - obj_next_data[0]),
                                -(obj_data[1] - obj_next_data[1])],
                        'dist_right': [-(obj_data[0] - obj_before_data[4]),
                                 -(obj_data[1] - obj_before_data[5])], 'height': obj_data[2],
                        'width': obj_data[3], 'height_left': obj_next_data[2],
                        'width_left': obj_next_data[3], 'height_right': obj_before_data[2],
                        'width_right': obj_before_data[3]}})

else:
   object_relations.update({obj: {'left': obj_next, 'right': obj_before,
                        'dist_left': [obj_data[0] - obj_next_data[0],
                                -obj_data[1] + obj_next_data[1]],
                        'dist_right': [-obj_data[0] + obj_before_data[0],
                                 -obj_data[1] + obj_before_data[1]], 'height': obj_data[2],
                        'width': obj_data[3], 'height_left': obj_next_data[2],
                        'width_left': obj_next_data[3], 'height_right': obj_before_data[2],
```

```
                    'width_right': obj_before_data[3]}})

time.sleep(10)
video.release()
cv.destroyAllWindows()
```

6.2.4. Praćenje gibajućeg objekta kamerom

Kada smo dobili potrebne podatke o referentnim objektima, njihovim međusobnim relacijama, njihovom redoslijedu u kojem su postavljeni u sobi, prelazi se na rješenje zadnjeg koraka, a to je izrada programskog koda pomoću kojeg će kamera pratiti gibajući objekt. Ovakav program će istovremeno morati izvršavati 2 procesa. Jedan proces će uzimati sliku sa kamere, obrađivati sliku, detektirati kretanje i računati potrebne udaljenosti. Spojila su se rješenja prijašnjih problema u jedan kod. Drugi proces će morati provjeravati izračunate udaljenosti gibajućeg objekta u odnosu na središte slike kamere. Ukoliko se gibajući objekt nalazi van predviđenih tolerancija program će slati naredbe kameri da se giba u potrebnom smjeru ili da se kamera zaustavi ukoliko se objekt nađe unutar tolerancija. Unutar prvog procesa, koji je zaslužan za obradu slike, bilo je potrebno riješiti nekoliko problema.

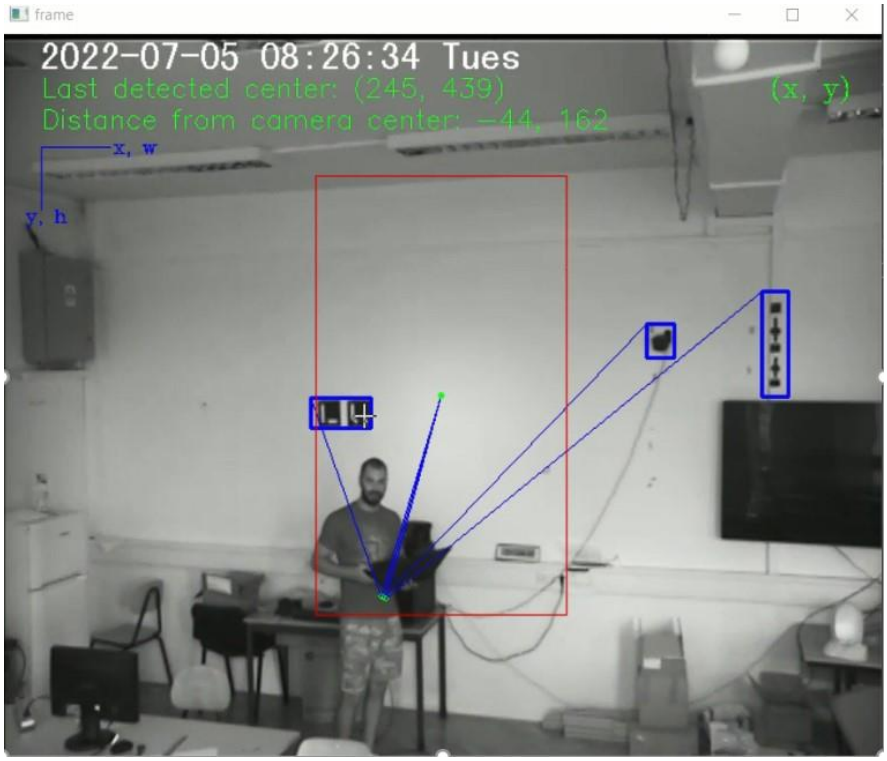
Prvi je problem nastajao prilikom traženja referentnih objekata dok se kamera pomicala. U određenim trenucima je program znao referentni objekt naći na krivom mjestu. To se uklonilo pomoću dobivenih podataka iz programa koji se izvršava prije glavnog programa. Iskoristili su se podaci relativnih međusobnih udaljenosti referentnih objekata te njihove minimalne vjerojatnosti identifikacije prilikom nalaženja na slici. Na početku programa kamera se postavlja u određenu *preset* poziciju u kojoj je kamera usmjerena prema ulazu u laboratorij. U programu je upisano koja se referentna točka nalazi u početnom *presetu*. S obzirom da znamo redoslijed predmeta u prostoriji, program učita podatke vezane za taj referentni objekt, vidi koji su mu susjedi te traži susjedne objekte na slici. Ukoliko ih nađe slijedi prva provjera – vjerojatnost identifikacije funkcije *match template*. Ukoliko su podaci ispod minimalne granice objekt se odbacuje. Ukoliko je vjerojatnost u odgovarajućim granicama, radi se druga provjera. Druga provjera se temelji na računanju udaljenosti susjednih referentnih objekata te usporedbom dobivenih rezultata sa prethodno dobivenim rezultatima iz prijašnjeg programa. Ovaj dio sa sigurnošću uklanja krivo nađene objekte ukoliko se dogodi da takva greška prođe prvu provjeru.

Drugi problem je nastajao prilikom praćenja gibajućeg objekta. Dok se kamera pomicala, program bi i dalje tražio razlike između prve i pete slike za svaki vremenski interval. S obzirom da se kamera pomicala, okoliš prve i pete slike je bio drukčiji. To bi nakon korištenja funkcije *absdiff* rezultiralo detekcijom promjene cijele slike. S obzirom da u tom slučaju nije moguće pronaći gibajući objekt na 'gibajućoj' slici, bilo je potrebno isključiti detekciju pronalaženja gibajućih objekata dok se kamera miče. Dok se kamera pomiče, program obustavlja detekciju gibanja, pomiče se na prethodno nađenu točku u prostoru koja je definirana udaljenošću od neke referentne točke. Ovdje se dolazi do trećeg problema.

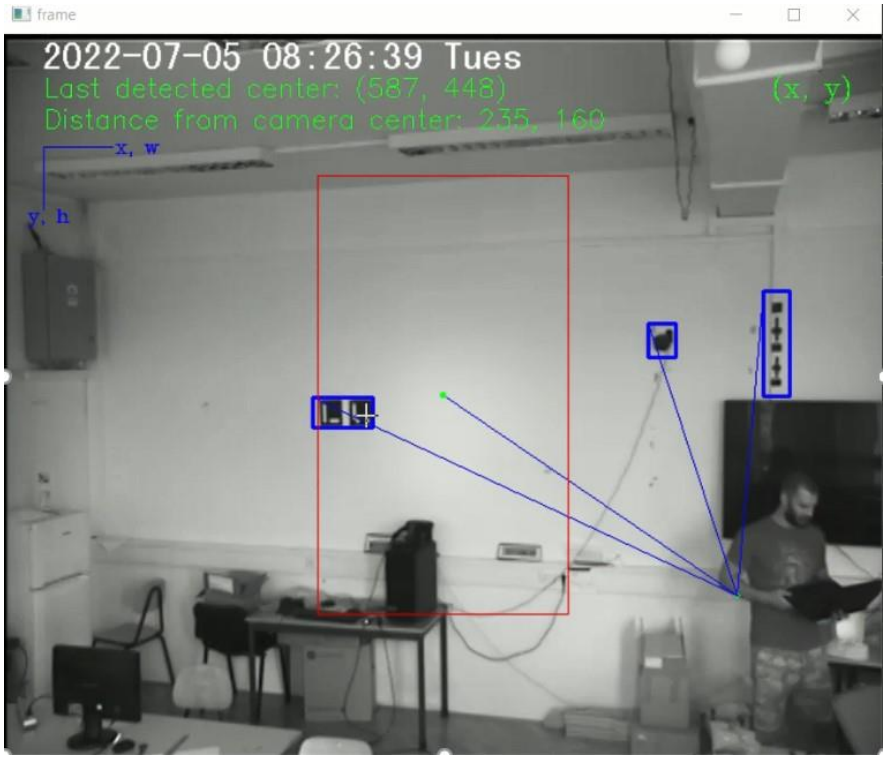
Treći problem je vezan uz kameru. Kamera u normalnim uvjetima prikazuje sliku u boji. Kamera također posjeduje rad u infracrvenom spektru. Iz nepoznatog razloga, vjerojatno

prilikom pomicanja kamere sa jedne lokaciju na drugu, moglo se dogoditi da je kamera bila udarena ili prilikom jačih vibracija su se spojevi elektronskih priključaka djelomično odspojili. Kamera radi isključivo u infracrvenom spektru, a do promjene dolazi nakon gibanja kamere. Nakon što kamera izvrši gibanje nekada se dogodi da se infracrveni spektar promjeni u sliku u boji i tada kamera zna ponovo fokusirati sliku. Ovdje dolazi do gubitka nađenih informacija na slici, odnosno gube se pronađene referentne točke i kamera izgubi trenutne podatke. Zbog ovog se uvelo kašnjenje nakon završetka gibanja kamere. Nakon što kamera završi gibanje i fokusira potrebnu točku u tolerancijsko polje, uvelo se kašnjenje ukoliko dođe do ponovnog fokusiranja slike i promjene boje koje traje 1-2 sekunde.

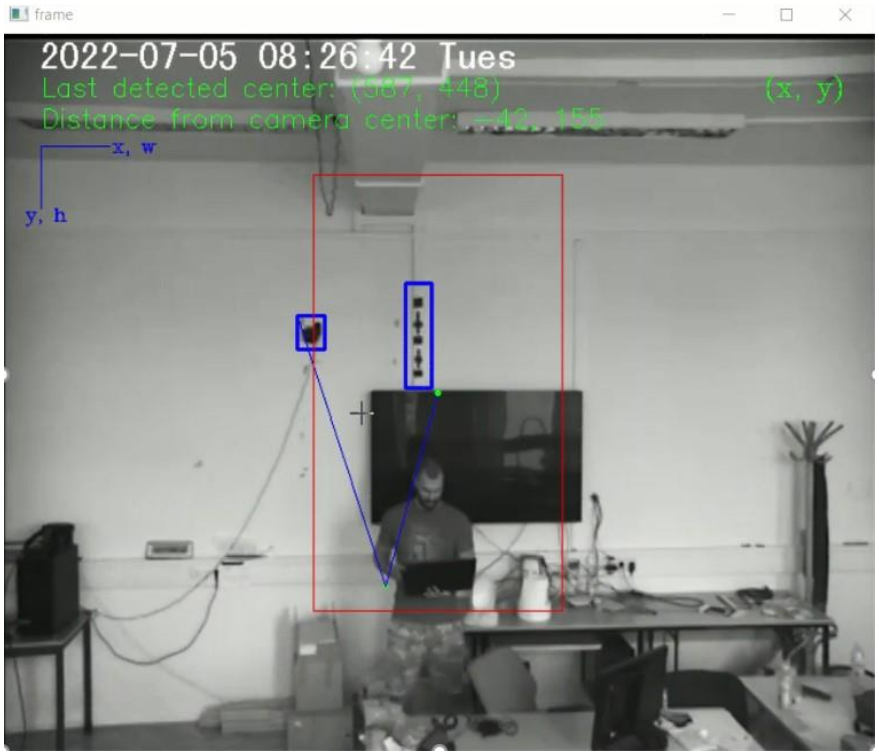
Program kronološki izvršava slijedeće funkcionalnosti. Ostvaruje se veza sa kamerom putem ONVIF standarda koji omogućuje kontroliranje kamere. Uzimaju se spremljene *preset* pozicije kako bi se kamera pri početku rada programa postavila u početni položaj i usmjerila prema ulazu u laboratorij. Nakon toga se uspostavlja nova veza sa kamerom preko *OpenCV* modula pomoću kojega program dohvaća i obrađuje sliku. Pokreće se drugi proces koji kontinuirano provjerava udaljenost nađenog objekta od središta slike kamere. Zatim se pokreće petlja unutar prvog procesa koja uzima sliku sa kamere. Unutar te petlje se nalazi kod za dohvaćanje slike te obradu sliku. Na dobivenoj slici sa kamere se prvo identificiraju referentni objekti te se radi se provjera nađenih referentnih objekata i njihovih međusobnih udaljenosti. Predviđaju se referentni objekti van kadra koji bi se mogli pojaviti ukoliko se kamera pomakne u obje strane kako bi program znao koje referentne točke mora tražiti s ciljem redukcije pogrešaka. Nakon provjere identificiranih objekata program analizira sliku, na prethodno navedeni način, kako bi našao gibajuće objekte. Ukoliko je identificiran objekt koji se pomaknuo računaju se udaljenosti objekta od središta slike kamere. U pozadini se odvija drugi proces koji izračunate vrijednosti analizira te zaključuje treba li kameru pomicati, a ako treba zaključuje u kojem smjeru i kada da stane. Ukoliko se kamera krene gibati prema nađenom objektu program prethodno zapamti točku u prostoru na kojem se objekt nalazio nakon odrađenog gibanja. Kada se kamera giba program i dalje traži referentni objekte na slici kako bi u svakom trenutku znao točku u koju se mora pomaknuti, ali ne traži gibanje zbog prethodno spomenutog problema gibajuće kamere. Nakon što kamera završi gibanje pokreće se odbrojavanje koje uklanja problem rekonfiguracije slike kamere. Kada odbrojavanje dođe do kraja, ponovo se detektira gibanje. Tada cijeli proces kreće ponovo te na tom principu funkcionira program. Slike praćenja gibanja su dane ispod kao i programski kod.



Slika 62 - Gibanje unutar tolerancijskog polja kamere



Slika 63 - Objekt se pomaknuo van tolerancijskog polja



Slika 64 - Vraćanje objekta u fokus kamere

Sa prikazanih slika vidimo kako kamera za svako detektirano gibanje računa udaljenosti od središta gibajućeg objekta do svakog referentnog objekta unutar kamere te samog središta slike. Na Slika 62. se čovjek nalazi unutar predviđenog tolerancijskog polja kamere. Na idućoj slici čovjek izlazi iz tolerancijskog polja. Sa slike se da očitati parametar '*Distance from frame center'* koji opisuje udaljenost objekta od središta kamere. Parametri pokazuju udaljenosti u odnosu na prikazan koordinatni sustav na slici. Tako se da zaključiti da se kamera mora gibati u desno kako bi objekt došao unutar tolerancijskog polja. Na Slika 64. vidimo ponovo kako je kamera postavila objekt u središte slike, a vrijednosti prethodnog parametra su unutar granica. Vrijednosti tolerancijskog polja su $\pm$100 px po x osi, a po y osi $\pm$175 px. Tolerancijsko polje je prikazano crvenim pravokutnikom na slici.

Kod:

```
import threading
import time
import cv2 as cv
import numpy as np
import os
from onvif import ONVIFCamera

# camera movement controls
cam = ONVIFCamera('192.168.0.200', 80, 'admin', 'admin')
media = cam.create_media_service()
ptz = cam.create_ptz_service()
profile = media.GetProfiles()[0]

frame_count = 0
switch = 0
img_before = None
img_now = None
center_roi_x = 100
center_roi_y = 175
moving_object_center = None
biggest_moving_object_center = None
moving_point_distance_x = 0
moving_point_distance_y = 0
object_relations = {
   'tape1.jpg': {'left': 'tape_door.jpg', 'right': None, 'dist_left': [-492, -23], 'dist_right': None, 'height': 53,
            'width': 22, 'height_left': 22, 'width_left': 45, 'height_right': None, 'width_right': None},
   'tape_door.jpg': {'left': 'hanger.jpg', 'right': 'tape1.jpg', 'dist_left': [-152, 37], 'dist_right': [492, 23],
              'height': 22, 'width': 45, 'height_left': 27, 'width_left': 65, 'height_right': 53,
              'width_right': 22},
   'hanger.jpg': {'left': 'tape_tv.jpg', 'right': 'tape_door.jpg', 'dist_left': [-307, -102], 'dist_right': [152, -
37],
              'height': 27, 'width': 65, 'height_left': 84, 'width_left': 21, 'height_right': 22,
```

```
                'width_right': 45},
    'tape_tv.jpg': {'left': 'vr_sensor.jpg', 'right': 'hanger.jpg', 'dist_left': [-88, 26], 'dist_right': [307, 102],
                    'height': 84, 'width': 21, 'height_left': 27, 'width_left': 22, 'height_right': 27,
                    'width_right': 65},
    'vr_sensor.jpg': {'left': 'aruco.jpg', 'right': 'tape_tv.jpg', 'dist_left': [-269, 61], 'dist_right': [88, -26],
                      'height': 27, 'width': 22, 'height_left': 24, 'width_left': 48, 'height_right': 84,
                      'width_right': 21},
    'aruco.jpg': {'left': 'aruco2.jpg', 'right': 'vr_sensor.jpg', 'dist_left': [-420, 30], 'dist_right': [269, -61],
                  'height': 24, 'width': 48, 'height_left': 81, 'width_left': 56, 'height_right': 27,
                  'width_right': 22},
    'aruco2.jpg': {'left': 'vr_sensor2.jpg', 'right': 'aruco.jpg', 'dist_left': [-214, -89], 'dist_right': [420, -30],
                   'height': 81, 'width': 56, 'height_left': 36, 'width_left': 34, 'height_right': 24,
                   'width_right': 48},
    'vr_sensor2.jpg': {'left': 'tape_window.jpg', 'right': 'aruco2.jpg', 'dist_left': [-336, 116],
                       'dist_right': [214, 89], 'height': 36, 'width': 34, 'height_left': 46, 'width_left': 24,
                       'height_right': 81, 'width_right': 56},
    'tape_window.jpg': {'left': None, 'right': 'vr_sensor2.jpg', 'dist_left': None, 'dist_right': [336, -116],
                        'height': 46, 'width': 24, 'height_left': None, 'width_left': None, 'height_right': 36,
                        'width_right': 34}}

    object_order = ['tape_window.jpg', 'vr_sensor2.jpg', 'aruco2.jpg', 'electric_box.jpg', 'aruco.jpg',
'vr_sensor.jpg',
                    'tape_tv.jpg', 'hanger.jpg', 'tape_door.jpg', 'tape1.jpg']
    to_find = ['tape_door.jpg']
    cwd = os.getcwd()
    objects_dir = "/objects"
    objects = os.listdir(cwd + objects_dir)
    results_min = {"tape1.jpg": 0.80,
                   "tape_door.jpg": 0.80,
                   "hanger.jpg": 0.6,
                   "tape_tv.jpg": 0.80,
                   "vr_sensor.jpg": 0.80,
                   "aruco.jpg": 0.75,
```

```
            "electric_box.jpg": 0.80,
            "vr_sensor2.jpg": 0.80,
            "aruco2.jpg": 0.75,
            "tape_window.jpg": 0.79}

last_found_objects = []
distance_to_object = {}
point = None
distance_from_frame_center = (None, None)
active_thread = False
moving = False
make_delay = False
counter = 0
camera_speed = 0.5
camera_sleep_time = 0.4


def MoveLeft():
    contMove = ptz.create_type('ContinuousMove')
    contMove.ProfileToken = profile.Name
    contMove.Velocity.PanTilt._x = -camera_speed
    contMove.Velocity.PanTilt._y = 0.0
    ptz.ContinuousMove(contMove)
    print('Moving left')


def MoveRight():
    contMove = ptz.create_type('ContinuousMove')
    contMove.ProfileToken = profile.Name
    contMove.Velocity.PanTilt._x = camera_speed
    contMove.Velocity.PanTilt._y = 0.0
    ptz.ContinuousMove(contMove)
    print('Moving right')
```

```
def MoveUp():
    contMove = ptz.create_type('ContinuousMove')
    contMove.ProfileToken = profile.Name
    contMove.Velocity.PanTilt._x = 0.0
    contMove.Velocity.PanTilt._y = camera_speed
    ptz.ContinuousMove(contMove)
    print('Moving up')


def MoveDown():
    contMove = ptz.create_type('ContinuousMove')
    contMove.ProfileToken = profile.Name
    contMove.Velocity.PanTilt._x = 0.0
    contMove.Velocity.PanTilt._y = -camera_speed
    ptz.ContinuousMove(contMove)
    print('Moving down')


def Stop():
    stop = ptz.create_type('Stop')
    stop.ProfileToken = profile.Name
    ptz.Stop(stop)


# camera movement function for a thread
def camera_move():
    global moving
    global make_delay
    global distance_from_frame_center
    global point
    global camera_sleep_time
```

```
global camera_speed

while True:
    time.sleep(0.5)
    if distance_from_frame_center is not None:
        try:

            if 100 < abs(distance_from_frame_center[0]) < 150:
                camera_speed = 0.3

            elif 150 < abs(distance_from_frame_center[0]) < 200:
                camera_speed = 0.4

            elif 200 < abs(distance_from_frame_center[0]) < 250:
                camera_speed = 0.5

            elif 250 < abs(distance_from_frame_center[0]) < 302:
                camera_speed = 0.6



        except:
            pass

        try:
            if distance_from_frame_center[0] < -center_roi_x:
                moving = True
                make_delay = True
                MoveLeft()
                time.sleep(camera_sleep_time)
                Stop()
                moving = False

            elif distance_from_frame_center[0] > center_roi_x:
```

```
                moving = True
                make_delay = True
                MoveRight()
                time.sleep(camera_sleep_time)
                Stop()
                moving = False

            if distance_from_frame_center[1] > center_roi_y:
                moving = True
                make_delay = True
                camera_speed = 0.3
                MoveDown()
                time.sleep(camera_sleep_time)
                Stop()
                moving = False

            if distance_from_frame_center is None:
                Stop()
        except:
            pass

        else:
            distance_from_frame_center = (None, None)
    cv.putText(frame2, f'camera speed: {camera_speed}',(150, 500), cv.FONT_HERSHEY_COMPLEX, 0.6,
150, 1)

  get_presets = ptz.create_type('GetPresets')
  get_presets.ProfileToken = profile.Name
  presets = ptz.GetPresets(get_presets)

  go_to_preset = ptz.create_type('GotoPreset')
  go_to_preset.ProfileToken = profile.Name
  go_to_preset.PresetToken = presets[len(presets)-1]._token
```

```
ptz.GotoPreset(go_to_preset)
time.sleep(2)
print('trying to connect to the camera')
cam = cv.VideoCapture('rtsp://192.168.0.200:554/snl/live/1/1/Ux/sido=-Ux/sido=')
print('camera connected')
time.sleep(1)

camera_thread = threading.Thread(target=camera_move, daemon=True)
camera_thread.start()

while True:

    check, frame = cam.read()
    check2, frame2 = cam.read()
    height, width, layers = frame.shape
    frame_center = (int(width / 2), int(height / 2))
    img_diff = None

    # blurring img for reducing noise on the image
    gray_not_blur = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
    gray = cv.GaussianBlur(gray_not_blur, (21, 21), 0)
    to_find_for_loop = to_find[:]
    found_object_results = {}
    found_object_results_list = []
    biggestArea = 0

    if len(to_find_for_loop) == 0:
        to_find_for_loop = last_found_objects[:]
        distance_from_frame_center = (None, None)

    # detecting objects in the frame and storing results
    for i in range(0, len(to_find_for_loop)):
        obj = to_find_for_loop[i]
```

```
            obj_img = cv.imread(cwd + objects_dir + "/" + obj)
            height_obj, width_obj, layers_obj = obj_img.shape
            obj_img_gray = cv.cvtColor(obj_img, cv.COLOR_BGR2GRAY)
            res = cv.matchTemplate(obj_img_gray, gray_not_blur, cv.TM_CCOEFF_NORMED)
            min_val, max_val, min_loc, max_loc = cv.minMaxLoc(res)
            x_obj = max_loc[0]
            y_obj = max_loc[1]
            obj_min_max_val = results_min[obj]

            # filtering match template results and removing objects on the edge of the frame
            if max_val > obj_min_max_val and 48 < x_obj and x_obj + width_obj < width - 48 and 48 < y_obj
< height - 48:
                found_object_results.update({obj: [x_obj, y_obj, max_val]})
                found_object_results_list.append(obj)
                cv.rectangle(frame, (x_obj, y_obj), (x_obj + width_obj, y_obj + height_obj), 255, 2)
            else:
                try:
                    to_find.remove(obj)
                    distance_to_object.pop(obj)
                except:
                    pass

            cv.putText(frame2, f'{found_object_results}', (150, 150), cv.FONT_HERSHEY_COMPLEX, 0.4, 255,
2)

        # If there is only 1 object listed for finding, there is no neighbour for distance control
        if len(found_object_results_list) is 1:
            obj = found_object_results_list[0]
            left = object_relations[obj]['left']
            right = object_relations[obj]['right']
            left_dist = object_relations[obj]['dist_left']
            right_dist = object_relations[obj]['dist_right']
            left_width = object_relations[obj]['width_left']
            left_height = object_relations[obj]['height_left']
```

```
        right_height = object_relations[obj]['height_right']
        right_width = object_relations[obj]['width_right']
        x_obj = found_object_results[obj][0]
        y_obj = found_object_results[obj][1]

        # if obj is on the right edge, there is only left neighbour
        if left is not None:
            a = x_obj + left_dist[0]
            b = y_obj + left_dist[1]
            if a > 50 and b > 50:
                to_find.insert(0, left)

        # if obj is on the left edge, there is only right neighbour
        if right is not None:
            a = x_obj + right_dist[0]
            b = y_obj + right_dist[1]
            if a < width - 50 and b > 50:
                to_find.append(right)

    # if there are 2 objects to find, there is no middle one for controlling the ones on the edge
    elif len(found_object_results_list) is 2:

        for i in range(0, len(found_object_results_list) - 1):
            obj = found_object_results_list[i]
            left = object_relations[obj]['left']
            right = object_relations[obj]['right']
            left_dist = object_relations[obj]['dist_left']
            right_dist = object_relations[obj]['dist_right']
            left_width = object_relations[obj]['width_left']
            left_height = object_relations[obj]['height_left']
            right_height = object_relations[obj]['height_right']
            right_width = object_relations[obj]['width_right']
            x_obj = found_object_results[obj][0]
```

```
        y_obj = found_object_results[obj][1]

        # check distance between his neighbour
        try:
            obj_next = found_object_results_list[i + 1]
            x_obj_next = found_object_results_list[i + 1][0]
            y_obj_next = found_object_results_list[i + 1][1]
            if obj_next == left:
                if not -10 < abs(x_obj - x_obj_next) - abs(left_dist[0]) < 10 and -10 < abs(
                        y_obj - y_obj_next) - abs(left_dist[1]) < 10:
                    cv.putText(frame2, f'{obj} distance with left neighbour not correct', (200, 200),
                               cv.FONT_HERSHEY_COMPLEX, 0.5, 100)

            elif obj_next == right:
                if not -10 < abs(x_obj - x_obj_next) - abs(right_dist[0]) < 10 and -10 < abs(
                        y_obj - y_obj_next) - abs(right_dist[1]) < 10:
                    cv.putText(frame2, f'{obj} distance with right neighbour not correct', (200, 200),
                               cv.FONT_HERSHEY_COMPLEX, 0.5, 100)
        except:
            pass

        # adding left neighbours if its on the edge and his neigh is already not in the list
        if left is not None and left not in found_object_results_list:
            if x_obj + left_dist[0] > 50 and y_obj + left_dist[1] > 50:
                to_find.insert(0, left)

        if right is not None and right not in found_object_results_list:
            if x_obj + right_dist[0] + right_width < width - 50 and y_obj + right_dist[1] < 50:
                to_find.append(right)

    # if there are more objects, control the ones on the edge with the ones in the middle
    elif len(found_object_results_list) > 2:
```

```
        # loop through the ones in the middle, check distance with the outer ones
        for i in range(1, len(found_object_results_list) - 1):
            obj = found_object_results_list[i]
            left = object_relations[obj]['left']
            right = object_relations[obj]['right']
            left_dist = object_relations[obj]['dist_left']
            right_dist = object_relations[obj]['dist_right']
            left_width = object_relations[obj]['width_left']
            left_height = object_relations[obj]['height_left']
            right_height = object_relations[obj]['height_right']
            right_width = object_relations[obj]['width_right']
            x_obj = found_object_results[obj][0]
            y_obj = found_object_results[obj][1]

            # getting object on the left edge
            obj_before = found_object_results_list[i - 1]
            x_obj_before = found_object_results[obj_before][0]
            y_obj_before = found_object_results[obj_before][1]

            # getting object on the right edge
            obj_next = found_object_results_list[i + 1]
            x_obj_next = found_object_results[obj_next][0]
            y_obj_next = found_object_results[obj_next][1]

            # if distance to the left object is good, check for adding his neighbour, if dist not good, remove
it
            if obj_before == left:
                if -10 < abs(x_obj - x_obj_before) - abs(left_dist[0]) < 10 and -10 < abs(y_obj - y_obj_before)
- abs(left_dist[1]) < 10:

                    left_before = object_relations[obj]['left']
                    right_before = object_relations[obj]['right']
                    left_dist_before = object_relations[obj]['dist_left']
                    right_dist_before = object_relations[obj]['dist_right']
```

```
                left_width_before = object_relations[obj]['width_left']
                left_height_before = object_relations[obj]['height_left']
                right_height_before = object_relations[obj]['height_right']
                right_width_before = object_relations[obj]['width_right']

                if left_before is not None and left_before not in found_object_results_list:
                    if x_obj_before + left_dist_before[0] > 50 and y_obj_before + left_dist_before[1] > 50:
                        to_find.insert(0, left_before)

            else:
                to_find.remove(obj_before)
                try:
                    distance_to_object.pop(obj_before)
                except:
                    pass

            # same for objects on the right, add neigh if dist is good, if dist is not good remove it
            if obj_next == right:
                if -10 < abs(x_obj - x_obj_next) - abs(right_dist[0]) < 10 and -10 < abs(y_obj - y_obj_next) -
abs(right_dist[1]) < 10:

                left_next = object_relations[obj]['left']
                right_next = object_relations[obj]['right']
                left_dist_next = object_relations[obj]['dist_left']
                right_dist_next = object_relations[obj]['dist_right']
                left_width_next = object_relations[obj]['width_left']
                left_height_next = object_relations[obj]['height_left']
                right_height_next = object_relations[obj]['height_right']
                right_width_next = object_relations[obj]['width_right']

                if right_next is not None and right_next not in found_object_results_list:
                    if x_obj_next + right_dist_next[0] + right_width_next > 50 and y_obj_next +
right_dist_next[1] > 50:
                        to_find.append(right_next)
```

```
            else:
                to_find.remove(obj_next)
                try:
                    distance_to_object.pop(obj_next)
                except:
                    pass

    if len(to_find_for_loop) != 0:
        last_found_objects = to_find_for_loop[:]
        cv.putText(frame2, f'{last_found_objects}', (150, 200), cv.FONT_HERSHEY_COMPLEX, 0.5, 255, 1)

    # catching frame fps=1
    if switch == 1:
        img_before = gray

    # catching frame fps=5
    if switch == 5:

        img_now = gray
        switch = 0

        # getting motion in form of pixels, changing it to binary
        # getting them into a uniform area and making contours
        if moving is False:

            if make_delay is False:

                # getting motion in form of pixels, changing it to binary
                # getting them into a uniform area and making contours

                img_diff = cv.absdiff(img_now, img_before)
                motion_threshold = cv.threshold(img_diff, 5, 255, cv.THRESH_BINARY)[1]
```

```
            threshold_dilated = cv.dilate(motion_threshold, np.ones((5, 5), np.uint8), iterations=3)
            contours, hierarchy = cv.findContours(threshold_dilated, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)

            # filtering out smaller ones, getting the biggest
            cv.drawContours(img_diff, contours, -1, (255, 255, 255), 2)

            for cnt in contours:
               cntArea = cv.contourArea(cnt)

               if cntArea > 4000:
                  x, y, w, h = cv.boundingRect(cnt)

                  if h > w:
                     moving_object_center = (int(x + w / 2), int(y + h / 2))
                     cv.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

                     if cntArea > biggestArea:
                        biggestArea = cntArea
                        biggestContour = cnt
                        x, y, w, h = cv.boundingRect(cnt)
                        biggest_moving_object_center = (int(x + w / 2), int(y + h / 2))
                        distance_from_frame_center = ((biggest_moving_object_center[0] - frame_center[0]), (biggest_moving_object_center[1] - frame_center[1]))
            else:

               counter += 1
               if counter == 3:
                  make_delay = False
                  counter = 0

      for obj in found_object_results:

         x_obj = found_object_results[obj][0]
```

```
        y_obj = found_object_results[obj][1]

        if biggest_moving_object_center is not None:

            if biggestArea != 0:
                # udaljenost središta gibanja u odnosu na referentni objekt
                moving_point_distance_x = biggest_moving_object_center[0] - x_obj
                moving_point_distance_y = biggest_moving_object_center[1] - y_obj
                distance_to_object.update({obj: [moving_point_distance_x, moving_point_distance_y]})
                point = (moving_point_distance_x + x_obj, moving_point_distance_y + y_obj)
                distance_from_frame_center = (point[0] - frame_center[0], point[1] - frame_center[1])
                cv.line(frame, (x_obj, y_obj), point, (255, 0, 0), 1)
                cv.circle(frame, point, 2, (0, 255, 0), 1)
                cv.line(frame, frame_center, point, (255, 0, 0), 1)



            elif biggestArea == 0 and len(distance_to_object) != 0:
                try:
                    moving_point_distance_x = distance_to_object[obj][0]
                    moving_point_distance_y = distance_to_object[obj][1]
                    point = (moving_point_distance_x + x_obj, moving_point_distance_y + y_obj)
                    distance_from_frame_center = (point[0] - frame_center[0], point[1] - frame_center[1])
                    cv.line(frame, (x_obj, y_obj), point, (255, 0, 0), 1)
                    cv.circle(frame, point, 2, (0, 255, 0), 1)
                    cv.line(frame, frame_center, point, (255, 0, 0), 1)
                except:
                    pass

    switch = switch + 1
    cv.putText(frame2, f'point: {point},distfromfc: {distance_from_frame_center}', (150, 300),
               cv.FONT_HERSHEY_COMPLEX, 0.5, 100, 1)
    cv.putText(frame2, f'x_obj: {x_obj}, y_obj: {y_obj}', (150, 350),
               cv.FONT_HERSHEY_COMPLEX, 0.5, 100, 1)
```

```
        cv.putText(frame, f'Last detected center: {biggest_moving_object_center}', (33, 50), cv.FONT_HERSHEY_SIMPLEX, 0.7,
                   (0, 255, 0), 1)
        cv.putText(frame, f'Distance from camera center: {distance_from_frame_center[0]}, {distance_from_frame_center[1]}',
                   (33, 75), cv.FONT_HERSHEY_SIMPLEX,
                   0.7, (0, 255, 0), 1)
        cv.circle(frame, (int(width / 2), int(height / 2)), 1, (0, 255, 0), 2)
        cv.rectangle(frame, (frame_center[0] + center_roi_x, frame_center[1] + center_roi_y),
                     (frame_center[0] - center_roi_x, frame_center[1] - center_roi_y), (0, 0, 255), 1)
        cv.line(frame, (33, 90), (33, 140), (255, 0, 0), 1)
        cv.line(frame, (33, 90), (88, 90), (255, 0, 0), 1)
        cv.putText(frame, 'x, w', (90, 95), cv.FONT_HERSHEY_COMPLEX, 0.5, (255, 0, 0), 1)
        cv.putText(frame, 'y, h', (20, 150), cv.FONT_HERSHEY_COMPLEX, 0.5, (255, 0, 0), 1)
        cv.putText(frame, '(x, y)', (width - 90, 50), cv.FONT_HERSHEY_COMPLEX, 0.7, (0, 255, 0), 1)
        cv.putText(frame2, f'counter: {counter}', (150, 400), cv.FONT_HERSHEY_COMPLEX, 1, (0, 0, 250), 1)
        cv.putText(frame2, f'camera speed: {camera_speed}',(150, 500), cv.FONT_HERSHEY_COMPLEX, 1, 150, 1)
        cv.imshow('frame', frame)
        cv.imshow('frame2', frame2)
        key = cv.waitKey(1)
        path = "C:/Users/CAD/PycharmProjects/pythonProject2/screenshots"
        if key == ord('x'):
            break
    cam.release()
    cv.destroyAllWindows()
```

6.3. Nedostaci

Nedostaci ovog pristupa rješavanju problema su slijedeći. S obzirom na položaj kamere nemoguće je pokriti cijeli prostor referentnim objektima. To dovodi do nemogućnosti kamere da prati objekt u svim dijelovima prostorije. Kamera je smještena visoko te pod određenim kutom veći dio slike je pokriven podom. S obzirom da se u prostoriji nalaze klupe, računala, stolice i slični objekti koji se svakodnevno pomiču nije moguće sa lakoćom odrediti referentne točke na način da se u svakom položaju kamere nalazi jedan referentni objekt. Iz tog razloga su oznake postavljane i tražene po zidovima prostora. Ovaj bi se problem mogao riješiti postavljanjem kamere na neku pogodniju poziciju.

Drugi nedostatak je rekonfiguracija kamere nakon završetka gibanja. Zbog rekonfiguracije kamere se moralo uvesti kašnjenje koje utječe na sam proces nalaženja gibanja. Ukoliko se objekt giba većom brzinom može se dogoditi da će, zbog uvedenog kašnjenja, objekt izaći iz slike prije nego što program ponovno počne detektirati gibanje te ga izgubi iz fokusa. Ovaj bi se problem mogao riješiti drugim pristupom riješavanju rada. Mogla bi se kalibrirati kamera. Ovaj bi pristup riješio i prvi problem. Povezalo bi se gibanje kamere sa realnim prostorom. Trebalo bi se izmjeriti koliko se stupnjeva kamera zarotira u određenom smjeru pri određenoj brzini te za koliko piksela se pomakne središte kamere krajnje slike i početne slike. Tada bi se po iznosu udaljenosti objekta od središta slike znalo koliko se kamera mora gibati da objekt dođe u središte slike. Ovaj problem bi uklonio potrebu za pronalaskom referentnih objekata po prostoriji te bi kamera trebala biti u stanju pratiti objekt gdje god da se nalazi u prostoru. S obzirom da prilikom rekonfiguracije kamere nakon završetka gibanja dolazi do gubitka referentnih točaka, a ovaj pristup ne zahtijeva traženje referentnih točaka u prostoru, rekonfiguracija kamere ne bi imala utjecaja na rad programa.

7. Zaključak

Razvoj tehnologije i industrije je doveo do novih potreba, proizvoda, metoda i standardizacija određenih procesa. Jedan od takvih slučajeva je standardizacija upravljanja IP PTZ kamerama. Razvio se ONVIF standard koji omogućuje svakome lako i jednostavno upravljanje IP PTZ kamerama. Osim toga, razvila se nova grana programiranja, računalni vid. To je znanost koja se bavi analizom i izvlačenjem informacija sa slike. Korištenjem raznovrsnih metodi i funkcija koje su ugrađene unutar modula *OpenCV*, ovaj rad se bavi detekcijom gibanja u okolini slike učitane sa kamere. Primjenom određenih metoda lako se detektira gibanje te računaju potrebni parametri. Korištenjem ONVIF standarda za upravljanje kamere te dobivenih parametara pomoću tehnika računalnog vida moguće je napraviti algoritam kojim će kamera pratiti gibanje iz scene. U radu je opisana problematika koja se javljala tokom izrade krajnjeg rješenja te su dani programski kodovi za svaki pojedinačni riješeni problem te konačno rješenje zadatka. Osim samog rješenja detaljno je opisana teorija koja se nalazi iza algoritma kako bi čitatelj lako mogao shvatiti funkcionalnosti algoritma i što se zbiva u pozadini napisanog programa.

8. Literatura

[1] https://industrija40.hr/

[2] https://www.happtory.hr/post/industrija-4-0

[3] https://www.innovationext.in/industry-readiness.html

[4] Stipančić Tomislav, Vizijski sustavi - predavanja, Fakultet strojarstva i brodogradnje, Zagreb 2020.

[5] https://expertphotography.com/how-does-a-camera-work/

[6] https://peytonbarbara.wixsite.com/otr-senzori/senzori-u-fotoaparatima

[7] https://www.sttmedia.com/colormodel-hsv

[8] https://www.adobe.com/creativecloud/file-types/image.html

[9] https://www.imyfone.com/phone-data-transfer/comparison-iphone-heic-vs-jpeg-format/

[10] https://opencv.org/

[11] http://paulbourke.net/miscellaneous/imagefilter/

[12] https://en.wikipedia.org/wiki/Digital_image_processing

[13] https://www.javatpoint.com/dip-introduction-to-frequency-domain

[14] http://www.web3.lu/internet-topological-structure/

[15] https://www.hierarchystructure.com/internets-hierarchical-structure/

[16] https://antmedia.io/rtsp-explained-what-is-rtsp-how-it-works/

[17] Rastović Mirko, Sinkronizirano upravljanje IP PTZ kamerom temeljeno na semantičkim mrežnim uslugama – Diplomski rad, Fakultet strojatstva i brodogradnje, Sveučilište u Zagrebu, 2017.

[18] https://www.onvif.org/

[19] https://www.python.org/