Razvoj sustava s više dubinskih kamera koji generira skupove podataka za treniranje neuronskih mreža

Novak, Vlatko

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Vlatko Novak

Zagreb, 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentori:

Prof. dr. sc. Marko Švaco
Dr. sc. Filip Šuligoj

Student:

Vlatko Novak

Zagreb, 2022.

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

Zahvaljujem mentoru doc.dr.sc Marku Švaci i komentoru dr.sc Filipu Šuligoju na usmjeravanju, savjetima i pomoći prilikom izrade rada.

Najviše se zahvaljujem ocu Zlatku, majci Biserki i sestri Valentini na podršci tijekom studiranja.

Vlatko Novak

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: **VLATKO NOVAK** Mat. br.: 0035210206

Naslov rada na hrvatskom jeziku: **Razvoj sustava s više dubinskih kamera koji generira skupove podataka za treniranje neuronskih mreža**

Naslov rada na engleskom jeziku: **Development of a multiple depth-camera system for generating datasets to train neural networks**

Opis zadatka:

Sustavi opskrbe i distribucije proizvoda rade s velikim količinama raznih proizvoda, gdje se za detekciju proizvoda mogu koristiti 3D kamere. Jedno od rješenja za detekciju proizvoda je treniranje neuronskih mreža primjenom velikog broj 2D ili 3D snimaka proizvoda. Kvalitetan postupak treniranja omogućuje postizanje veće točnosti lokalizacije objekata ako je skup podataka za treniranje kvalitetno obrađen, te je stoga glavni zadatak ovog rada razviti sustav za automatsko generiranja skupova podataka o proizvodima namijenjenog treniranju neuronskih mreža.
U sklopu rada potrebno je:

- Projektirati i izraditi laboratorijski postav koji se sastoji od tri dubinske kamere usmjerene na područje interesa,
- Razraditi program i postupak intrinzične i ekstrinzične kalibracije svih kamera,
- Programirati c++ aplikaciju koja sinkronizirano dohvaća 2D slike i 3D oblake točaka,
- Procesirati podatke s kamera tako da se automatski izvrši: segmentacija proizvoda u 2D slikama, rekonstrukcija 3D geometrije proizvoda pomoću kalibriranih transformacija između kamera, spremanje konačnih verzija skupova podataka,
- Testirati i evaluirati kvalitetu i primjenjivost podataka za treniranje neuronskih mreža namijenjenih za detekciju objekata.

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

Zadatak zadan:	Rok predaje rada:	Predvideni datum obrane:
5. svibnja 2022.	7. srpnja 2022.	18. srpnja do 22. srpnja 2022.
Zadatak zadao:	Komentor:	Predsjednica Povjerenstva:
doc. dr. sc. Marko Švaco	dr. sc. Filip Šuligoj	prof. dr. sc. Biserka Runje

SADRŽAJ

1. UVOD ... 1
1.1. Problematika i pristup zadatku ... 1
2. DUBINSKE KAMERE ... 3
2.1. Princip rada stereo kamere ... 3
2.2. Podjela stereo vizije ... 5
2.2.1. Pasivna stereo vizija ... 5
2.2.2. Aktivna stereo vizija ... 6
2.3. Intel RealSense D435 kamera ... 6
3. PROJEKTIRANJE POSTAVA I KONSTRUIRANJE NOSAČA ZA KAMERU ... 10
3.1. Konstrukcija nosača kamere ... 11
3.2. Simulacija postava ... 13
3.3. Razvoj postava u laboratoriju ... 16
4. INTRINZIČNA KALIBRACIJA KAMERE ... 19
4.1. Geometrija formiranja slike ... 19
4.1.1. Homogene koordinate ... 23
4.2. Kalibracija kamere pomoću OpenCV ... 24
4.2.1. Proces uzimanja slika za intrinzičnu kalibraciju kamere ... 25
5. EKTRINZIČNA (STEREO) KALIBRACIJA KAMERE ... 31
5.1. Spremanje dobivenih transformacija ... 36
6. REKONSTRUKCIJA 3D GEOMETRIJE PROIZVODA ... 38
6.1. Oblak točaka (eng. Point Cloud) ... 39
6.2. Veličina oblaka točaka ... 40
6.2.1. Smanjenje uzorkovanja (Voxel downsampling) ... 41
6.3. Segmentacija pozadine u oblaku točaka ... 44
6.3.1. Segmentacija žutih točaka (SelectedByIndex) ... 44
6.3.2. Izrezivanje oblaka točaka (Crop) ... 46
6.4. Primjena stereo kalibracije na oblake točaka ... 48
6.5. Primjena ICP na oblake točaka ... 52
6.6. Rekonstrukcija 3D geometrije proizvoda ... 56
7. GENERIRANJE SETA 2D FOTOGRAFIJA ZA TRENIRANJE NEURONSKIH MREŽA ... 60
7.1. Manualno generiranje ... 60
7.2. Automatsko generiranje ... 62
7.2.1. Generiranje velikog broja slika ... 62
7.2.2. Funkcija inRange() ... 63
7.2.2.1. 3D segmentacija – dobivanje potrebnih indeksa ... 66
7.2.3. Pronalazak i označavanje kontura ... 67
7.3. YOLO struktura direktorija ... 69
7.3.1. Problem zapisa koordinata oznake predmeta ... 70
8. TESTIRANJE I EVALUACIJA PRIMJENJIVOSTI PODATAKA ZA TRENIRANJE NEURONSKIH MREŽA ... 73

8.1. Funkcioniranje konvolucijskih neuronskih mreža 73
8.2. YOLOv5 75
8.2.1. Korištenje YOLOv5 algoritma na Google Colab-u 76
8.3. Evaluacija rezultata 78
9. ZAKLJUČAK 81

POPIS SLIKA

Slika 1. Jednostavni stereo sistem – izometrijski pogled [1] ... 3
Slika 2. Jednostavni stereo sistem – tlocrt [1] ... 4
Slika 3. Stereo disparitet [1] ... 5
Slika 4. Pasivna stereo vizija [2] ... 5
Slika 5. Aktivna stereo vizija [2] ... 6
Slika 6. Intel RealSense D435 kamera [3] ... 7
Slika 7. Rolling i Total Shutter princip [4] ... 8
Slika 8. Laboratorijski postav – prva verzija ... 10
Slika 9. Neke konfiguracije Intel-ovih kamera [5] ... 11
Slika 10. Vijak za alu-profil ... 11
Slika 11. Vijak DIN 7991 M3 ... 12
Slika 12. Kamera s dimenzijama provrta ... 12
Slika 13. Nosač kamere u CATIA-i ... 13
Slika 14. Simulacija postava – izometrijski pogled ... 14
Slika 15. Simulacija postava – pogled od naprijed ... 15
Slika 16. Simulacija postava – pogled odozgo ... 15
Slika 17. Laboratorijski postav – kamere montirane ... 16
Slika 18. Laboratorijski postav – prikaz 1 ... 17
Slika 19. Laboratorijski postav - slika 2 ... 17
Slika 20. Laboratorijski postav – slika 3 ... 18
Slika 21. Pinhole model kamere [6] ... 20
Slika 22. Oblici distorzije [6] ... 23
Slika 23. Dijagram toka intrinzične kalibracije kamere ... 24
Slika 24. Šahovska ploča za kalibraciju kamere ... 25
Slika 25. Primjer uzimanja fotografija za kalibraciju ... 26
Slika 26. Ograničenje kuta kod uzimanja fotografija za kalibraciju [7] ... 26
Slika 27. Fotografije uzete za kalibraciju jedne kamere ... 27
Slika 28. Snippet za kalibraciju kamere ... 28
Slika 29. Prikaz pravilno i nepravilno pronađene koordinatne osi na fotografiji ... 29
Slika 30. Snippet iscrtavanje koordinatne osi i pronađenih kuteva na fotografiji ... 30
Slika 31. Teorija stereo kalibracije kamera [8] ... 31
Slika 32. Snippet sintaksa stereo kalibracijske funkcije ... 31
Slika 33. Dijagram toka stereo kalibracije ... 32
Slika 34. Primjer slika uzetih za stereo kalibraciju ... 34
Slika 35. Podaci stereo kalibracije kamera 1-2 ... 35
Slika 36. Podaci stereo kalibracije kamera 1-3 ... 35
Slika 37. Snippet spremanja podataka u .yaml datoteku ... 36
Slika 38. Primjer YAML zapisa intrinzičnih parametara kamere ... 36
Slika 39. Primjer YAML zapisa ekstrinzičnih parametara kamera jedan i dva ... 37
Slika 40. Snippet čitanja podataka spremljenih u .yaml datoteku ... 37
Slika 41. Struktura oblaka točaka [9] ... 39
Slika 42. Primjer oblaka točaka [10] ... 39
Slika 43. Izgled pohranjenog oblaka točaka ... 40
Slika 44. Veličina oblaka točaka ... 41
Slika 45. Snippet sintaksa *VoxelDownSample()* ... 41

Slika 46. VoxelDownSample – 1 42
Slika 47. VoxelDownSample – 2 42
Slika 48. VoxelDownSample – 3 43
Slika 49. 2D fotografija s pripadajućim 3D oblakom točaka 44
Slika 50. Piksel 2D [11] 45
Slika 51. Snippet sintaksa *SelectByIndex()* 45
Slika 52. Segmentirani oblak točaka 45
Slika 53. Snippet funkcije *Crop()* 46
Slika 54. Oblak točaka nakon izrezivanja 47
Slika 55. 'Timeline' jednog oblaka točaka – priprema za ICP 48
Slika 56. 2D prikaz Cedevite sa svih kamera 49
Slika 57. Oblaci točaka sa sve tri kamere prikazani zajedno 49
Slika 58. Snippet čitanja i upisivanja transformacijske matrice u željenu varijablu 50
Slika 59. Transformacijska matrica kamere 2 na kameru 1 50
Slika 60. Transformacijska matrica kamere 3 na kameru 1 50
Slika 61. Snippet transformacija oblaka točaka 50
Slika 62. Primjer Cedevita nakon stereo kalibracije 51
Slika 63. Intel kutija prije stereo kalibracije 51
Slika 64. Intel kutija nakon stereo kalibracije 52
Slika 65. Snippet ICP 54
Slika 66. Cedevita nakon primjene ICP registracije 56
Slika 67. Snippet postupka za 3D rekonstrukciju geometrije proizvoda 57
Slika 68. Rekonstruirana 3D geometrija proizvoda Cedevite 59
Slika 69. Manualno označavanje predmeta 60
Slika 70. Mogućnosti augmentacije u roboflow 61
Slika 71. Manualno označeni set slika spreman za treniranje neuronske mreže 62
Slika 72. Snippet generiranje velikog broja slika 63
Slika 73. Snippet *inRange()* 64
Slika 74. Početna BGR slika 64
Slika 75. HSV slika 64
Slika 76. Izlaz *inRange()* funkcije 65
Slika 77. Slika nakon *bitwise_not()* 65
Slika 78. Rezultat - boja segmentirana 66
Slika 79. Snippet koda za vektor izbačenih indeksa 67
Slika 80. Primjer pronalaska i iscrtavanja kontura 67
Slika 81. Snippet *findContours()* funkcije 68
Slika 82. Snippet *rectangle()* funkcije 68
Slika 83. Označen željeni predmet 69
Slika 84. Označeni željeni predmet na početnoj slici 69
Slika 85. Struktura YOLOv5 direktorija [13] 70
Slika 86. Primjer zapisa koordinata pravokutnika OpenCV vs Yolo 70
Slika 87. OpenCV kote pravokutnika 71
Slika 88. YOLO kote pravokutnika 71
Slika 89. Snippet pretvorba koordinata pravokutnika za YOLO 72
Slika 90. Snippet pohrana tekstualne datoteke 72
Slika 91. Balansirani set za treniranje neuronske mreže 73
Slika 92. Matematički prikaz konvolucije [13] 74
Slika 93. Matematički prikaz Max pooling 2x2 [14] 74
Slika 94. Primjer konvolucije [14] 75
Slika 95. Primjer Max pooling [14] 75

Slika 96. YOLOv5 [15] 76
Slika 97. YOLOv5 Google Colab početak 77
Slika 98. YOLOv5 početak treninga 77
Slika 99. YOLOv5 datoteka s klasama i putanjama 77
Slika 100. YOLOv5 testiranje 78
Slika 101. Rezultati detekcije objekata na testnim slikama 79
Slika 102. Analiza treniranja neuronske mreže 80

POPIS OZNAKA

Oznaka	Mjerna jedinica	Opis oznake
x_l		X koordinata lijeve kamere
y_l		Y koordinata lijeve kamere
f, f_x, f_y	mm	Žarišna duljina kamere
X		X koordinata točke P
Y		Y koordinata točke P
Z		Z koordinata točke P
x_r		X koordinata desne kamere
y_r		Y koordinata desne kamere
T_x	mm	Udaljenost između kamera
d	mm	Stereo disperitet
p		2D piksel na ravnini slike
K		Intrinzična matrica kamere
R		Rotacija koordinatnog sustava kamere
t		Translacija koordinatnog sustava kamere
P_w		3D točka izražena u odnosu na globalni koordinatni sustav
P_c		3D točka izražena u odnosu na kamerin koordinatni sustav
c_x, c_y		Udaljenost senzora kamere od optičkog centra
x', y'		Normalizirane koordinate kamere
X_c, Y_c, Z_c		Koordinate 3D točke predstavljene u koordinatama kamere
X_w, Y_w, Z_w		Koordinate 3D točke predstavljene u globalnim koordinatama
k_1, k_2, k_3, k_4, k_5		Distorzijski parametri (radijalni koeficijenti)
p_1, p_2		Distorzijski parametri (tangencijalni koeficijenti)
s_1, s_2, s_3, s_4, s_5		Distorzijski parametri (koeficijenti distorzije tanke prizme)
u, v		Koordinate 2D piskela na ravnini slike

POPIS KRATICA

Kratica		**Opis**
CAD		*Computer Aided Design* – računalom potpomognuto oblikovanje
ISO		*International organization for standardization* – Međunarodna organizacija za standardizaciju
ICP		Iterative Closest Point – iteracija najbližih točaka
IDE		Integrated Development Environment – integrirano razvojno okruženje

SAŽETAK

U ovom radu je izrađen sustav za rekonstrukciju 3D geometrije proizvoda i za automatsko generiranje seta 2D slika spremnih za treniranje neuronskih mreža. Prvo je simuliran postav s 3 kamere i konstruiran nosač za kameru u programskom CAD paketu CATIA V5R20. Nosač za kamere je zatim napravljen aditivnom tehnologijom 3D printanja. Korišten je postav od aluminijskih profila na kojima su bile montirane 3 kamere Intel Realsense D435. Kamerama se upravljalo kodom u C++ programskom jeziku pisanim u integriranom razvojnom okruženju (engl. IDE) QT. Korištene su biblioteke: Open3D koja služi za dohvaćanje i obradu oblaka točaka, Eigen za operacije s linearnom algebrom, OpenCV za 2D obradu slika, te biblioteke dane od strane proizvođača kamere pod nazivom *Librealsense* za konfiguriranje i upravljanje kamerama. Za 3D rekonstrukciju predmeta korišteni su koncepti stereo kalibracije kamera, smanjenje uzorka oblaka točaka, izrezivanje, odabir i izbacivanje po indeksu oblaka točaka te algoritam za minimiziranje razlike između dva oblaka točaka ICP (*Iterative Closest Point*). Za generiranje 2D slika spremnih za treniranje neuronskih mreža koristile su se mnoge funkcije dostupne u biblioteci OpenCV. Neke od funkcija su: pretvorba slike u boji u slike u nijansama sive, pretvorba u binarnu sliku, segmentacija boje iz slike, pronalazak kontura na slici, iscrtavanje pravokutnika oko predmeta. Nakon što je generiran set slika, one su dane neuronskoj mreži YOLOv5 na treniranje kako bi neuronska mreža bila spremna za kasniju detekciju predmeta. YOLOv5 je pokazao visok postatak uspješnosti detekcije naučenih predmeta - točnost od oko 90%.

Ključne riječi: stereo kamera, 3D rekonstrukcija geometrije proizvoda, automatsko generiranje slika za treniranje neuronske mreže, C++, Open3D, Eigen, OpenCV, ICP, YOLOv5, detekcija predmeta.

SUMMARY

In this thesis, a system for 3D reconstruction of product's geometry and automatically generated dataset of 2D images fit to train neural networks was developed. Firstly, a setup with three cameras was simulated, and a camera carrier was design in software package CATIA V5R20. The camera carrier was made via 3D printing additive manufacturing process. The setup was made out of aluminum profiles, on top of which three Intel Realsense D435 cameras were montaged. Cameras were managed by a code written in C++ programming language in integrated development environment (IDE) QT. There were used libraries: Open3D for capturing and processing point cloud, Eigen for operations with linear algebra, OpenCV for 2D image processing and libraries supplied by camera manufacturer named *Liberalsense* for configuring and managing cameras. Concepts like stereo camera calibration, reducing the number of points, cropping, selecting by indices and algorithm for minimizing the difference between two point clouds ICP (*Iterative Closest Point*) were used for 3D reconstruction of product's geometry. A lot of functions from an OpenCV library were used for generating 2D images ready to train a neural network. Some used functions are: color image conversion to *grayscale*, conversion to binary image color segmentation, contours detection, drawing a rectangle on image. After the dataset of images is generated, they are forwarded to neural network YOLOv5 for training so that the neural network is ready for object detection. YOLOv5 showed a high rate of detection of learned objects – an accuracy of about 90%.

Key words: stereo camera, 3D reconstruction of product's geometry, automatic generation of images for neural network training, C++, Open3D, Eigen, OpenCV, ICP, YOLOv5, object detection.

1. UVOD

Računalni vid znanstvena je i tehnološka disciplina koja se bavi teorijom i izradom samih sustava koji služe dobivanju informacija bilo to iz fotografija, video izradaka ili određenih medicinskih uređaja. Uz to cilj računalnog vida je prepoznavanje objekata, rekonstrukcija slike i sl. Ljudi su vizualna bića, najviše informacija dobivaju preko osjetila vida, tako i kod strojeva, vid im može dati širok spektar mogućnosti. Trend razvoja i korištenje računalnog vida u značajnom je porastu kako tehnologija teži industriji 4.0, jer je računalni vid važna komponenta u podizanju nivoa autonomnosti i inteligenciji strojeva. Pametnom implementacijom algoritama može se upravljati računalnim vidom na načine koji omogućuju strojevima i uređajima mogućnosti automatizacije raznih procesa. Teži se automatiziranoj ekstrakciji, analizi i razumijevanju korisnih podataka s pojedine slike ili većeg broja slika. Napretkom metoda dubokog učenja, pojavila se veća potreba za područjem računalnog vida, jer ova dva područja zajedno su veliki korak prema autonomnosti.

1.1. Problematika i pristup zadatku

Problematika ovog rada temelji se na osmišljavanju i programiranju rutina u C++ aplikaciji za dohvaćanje i obradu 2D slika i 3D oblaka točaka, kako bi se dobili set slika za treniranje neuronske mreže i mogućnost rekonstruiranja 3D geometrija proizvoda. Prvi problem je dohvaćanje 2D i 3D podataka koji se rješava provođenjem kalibracije i stereo kalibracije. Obje kalibracije su zaslužne za dohvaćanje što realnijih podataka scene i za preciznu 3D rekonstrukciju.

3D rekonstrukcija scene je moderni problem te je u ovom radu prikazana 3D rekonstrukcija geometrije proizvoda u novoj open-source knjižnici Open3D koja je još u razvoju i postaje sve popularnija zbog svoje jednostavne sintakse, te omogućava razne obrade i manipulacije nad oblakom točaka (engl. Point Cloud). Osmišljena je 3D rekonstrukcija pomoću stereo kalibracije, odnosno rotacije i translacije 3D podataka jedne kamere na drugu i zatim slijedi potpuno preklapanje oblaka točaka pomoću ICP registracije, koja dva bliska oblaka točaka preklopi po točkama koji su im isti.

Stvaranje seta fotografija za treniranje neuronskih mreža najčešće još uvijek nije automatizirani proces. Zasad se najčešće željeni predmeti prvo moraju manualno označiti na slikama te onda dati neuronskoj mreži na treniranje. U ovom radu je prikazan postupak kako

je u kontroliranim uvjetima moguće ostvariti automatsko generiranje seta označenih slika spremnih za treniranje neuronske mreže. Segmentacijom se uklanja pozadina i izoliraju se pikseli predmeta, koji se zatim mogu automatski označiti funkcijama iz OpenCV knjižnice, te se tako dobiju slike spremne za treniranje neuronske mreže.

U računalnom vidu najčešće se koriste konvolucijske neuronske mreže jer su najbolje za rad s ovom vrstom podataka kao što su fotografije. YOLO je najpoznatija neuronska mreža za učenje iz fotografija i zatim detekciju naučenih predmeta zato što je lako dostupna, brza i relativno laka za korištenje.

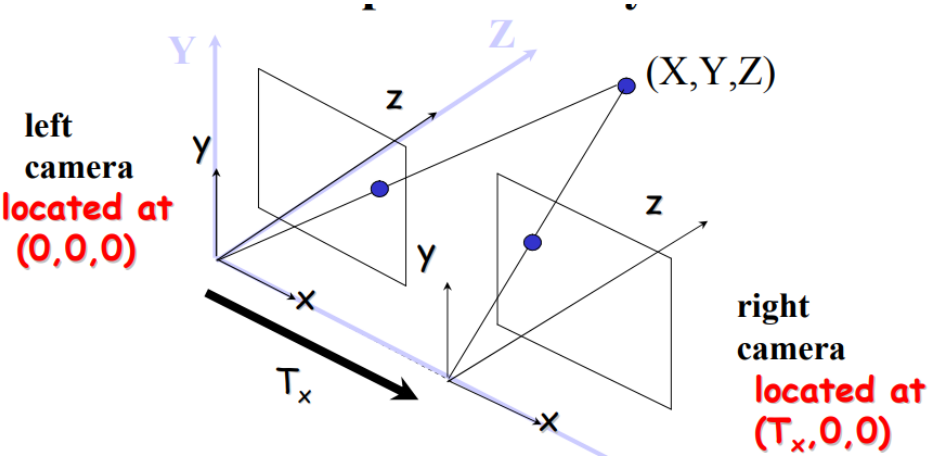
2. DUBINSKE KAMERE

Dubinske ili stereo kamere su kamere koje rješavaju problem percepcije dubine simulacijom ljudskog binokularnog vida. Binokularnim vidom ove kamere dobivaju sposobnost hvatanja trodimenzionalnih fotografija. Stereo vizijski sustavi imaju veliku važnost u područjima kao što je robotika, za izdvajanje informacije o relativnom položaju 3D objekata u autonomnim sustavima. Druga primjena u robotici bila bi detekciju objekata, gdje informacija o dubini omogućava sustavu da odvoji okluziju predmeta na slici, na primjer kada se jedan predmet nalazi ispred drugoga, u tom slučaju robot ne bi mogao razlikovati zasebne objekte ni po kojem drugom kriteriju.

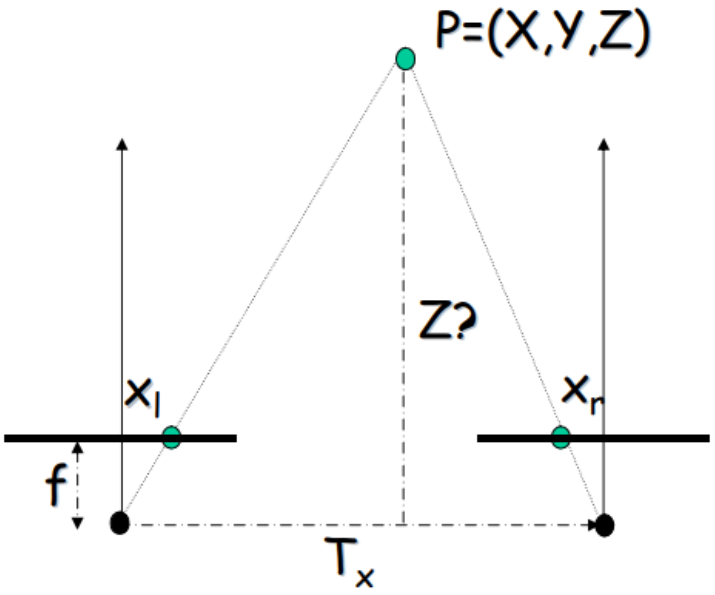
2.1. Princip rada stereo kamere

Ovdje će biti dani samo osnovni pojmovi i matematički izrazi, za više detalja pogledati reference [1] i [2]. Stereo vizija temelji se na triangulaciji zraka s više gledišta, točnije koriste se dva senzora na određenoj udaljenosti za triangulaciju sličnih piksela iz obje 2D ravnine. Udaljenost između leća kod tipične stereo kamere je aproksimativno udaljenost između ljudskih očiju koja iznosi oko 6.35 cm. Svaki piksel na slici digitalne kamere prikuplja svjetlost koja dopire do kamere duž 3D zraka. Ako se značajka u svijetu može identificirati kao mjesto piksela na slici, znamo da ta značajka leži na 3D zraci koja je povezana s tim pikselom. Ako koristimo više kamera možemo dobiti više zraka. Pronalaženje mjesta na kojem se te zrake sijeku govori nam o 3D lokaciji objekta i njegovim značajkama.

Za primjer uzimamo stereo kameru u kojoj su dvije kamere potpuno iste (ista orijentacija, iste žarišne duljine) samo je jedna pomaknuta po X osi za udaljenost *Tx* prikazano na slici 1 i 2.



Slika 1. Jednostavni stereo sistem – izometrijski pogled [1]



Slika 2. Jednostavni stereo sistem – tlocrt [1]

Koordinate slike točke P u lijevoj kameri su:

$$x_l = f\frac{X}{Z},\ y_l = f\frac{Y}{Z}. \quad \textbf{(1)}$$

Gdje je:

x_l	x koordinata lijeve kamere
y_l	y koordinata lijeve kamere
f	žarišna duljina kamere
X	X koordinata točke P
Y	Y koordinata točke P
Z	Z koordinata točke P (dubina).

Koordinate slike točke P u desnoj kameri su:

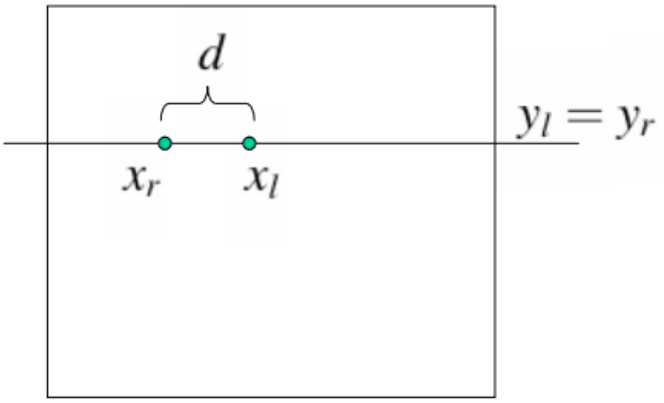
$$x_r = f\frac{X-T_x}{Z},\ y_r = f\frac{Y}{Z} \quad \textbf{(2)}$$

x_r	x koordinata desne kamere
y_r	y koordinata desne kamere
T_x	udaljenost između kamera.

Stereo disparitet se prikazan na slici 3 se zatim računa:

$$d = x_l - x_r = f\frac{X}{Z} - \left(f\frac{X}{Z} - f\frac{T_x}{Z}\right), \quad \textbf{(3)}$$

$$d = \frac{fT_x}{Z}. \quad \textbf{(4)}$$



Slika 3. Stereo disparitet [1]

Sada je jasno da je formula za dobivanje Z koordinate točke (dubine):

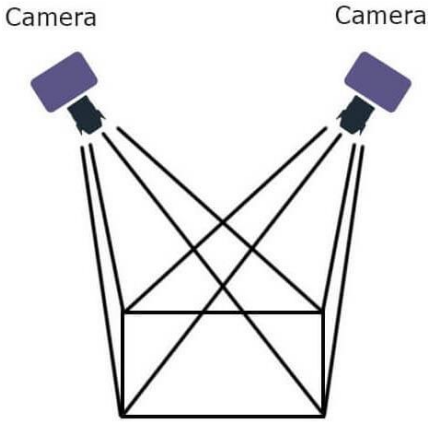
$$Z = \frac{fT_x}{d}. \quad \textbf{(5)}$$

2.2. Podjela stereo vizije

Kvaliteta rezultata koje kamerama primarno ovisi o gustoći vizualno prepoznatljivih točaka koje se podudaraju. Bilo koji izvor teksture – prirodni ili umjetni- značajno će poboljšati točnost. Zato je iznimno korisno imati opcijski projektor teksture koji obično može dodati detalje izvan vidljivog spektra. Osim toga, ovaj projektor se može koristiti kao umjetni izvor svjetla za noćne situacije.

2.2.1. Pasivna stereo vizija

Pasivni stereo sustav ovisi o dostupnom svjetlu iz okoline i ne koristi nikakvu vrstu vanjskog svjetla. Sustav prikazan na slici 4.



Slika 4. Pasivna stereo vizija [2]

Pasivni stereo sustavi pogodni su za dobro osvijetljene regije.

Prednosti pasivnog stereo sustava:

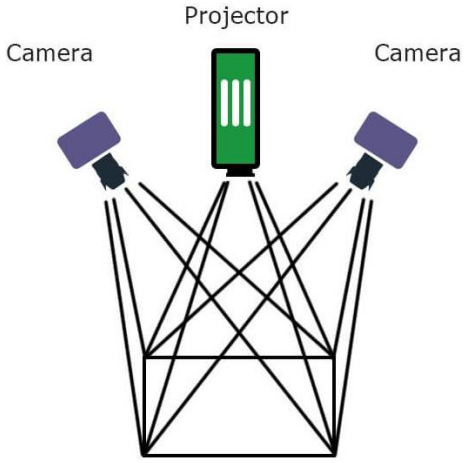
1. Dobro radi u okolini sa strukturiranim svjetlom ili na sunčevoj svjetlosti
2. Jeftiniji od aktivnih stereo sistema.

Mane pasivnog stereo sustava:

1. Prosječan rad pri slabom osvjetljenju
2. Prosječan rad u okolini bez strukture.

2.2.2. Aktivna stereo vizija

Aktivni stereo sustav je tip stereo vizije koji aktivno koristi svjetlo kao što je laser ili strukturirano svjetlo kako bi se pojednostavio problem stereo podudaranja. Ovaj sustav koristan je u regijama gdje nedostaje svjetla. Infracrveni projektor ili drugi izvor svjetlosti preplavit će scenu teksturom i time će prekinuti ovisnost o vanjskom izvoru svjetlosti. Sustav prikazan na slici 5.



Slika 5. Aktivna stereo vizija [2]

Prednosti aktivnog stereo sustava:

1. Dobro radi u okolini slabe svjetlosti
2. Dobro radi u unutarnjem prostoru bez strukturiranog svjetla.

Mane aktivnog stereo sustava:

1. U sceni sa sunčevim svjetlom, isti je rezultat kao kod pasivnog stereo sustava
2. Na većim udaljenostima isti je rezultat kao kod pasivnog stereo sustava
3. IR projektor povećava cijenu.

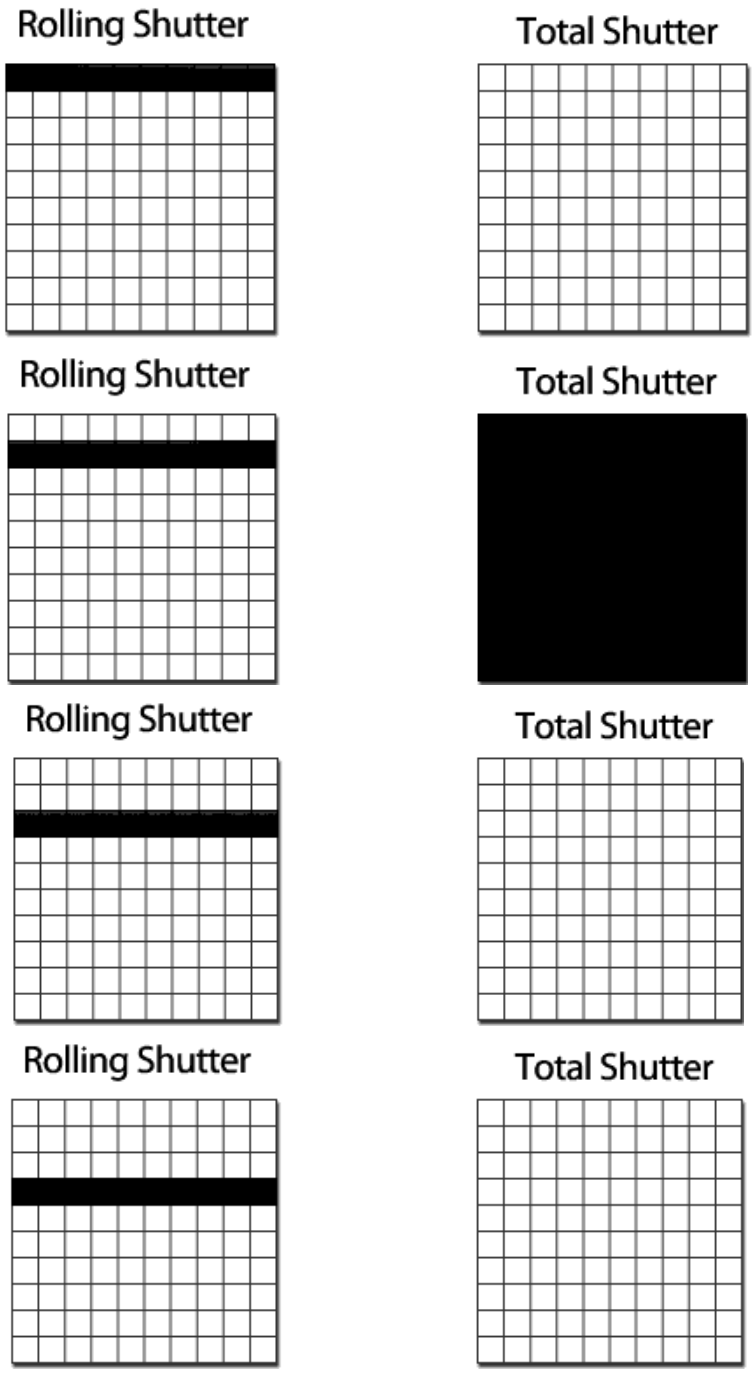
2.3. Intel RealSense D435 kamera

Oznaka D u imenu označava „Depth“. Intelove Realsense kamere serije D400 koriste stereo viziju kako bi proračunale dubinu. Cijela Intelova D400 serija pruža zanimljive kamere po

prihvatljivoj cijeni. Kamera D435 sadrži USB-C priključak, dva senzora dubine, RGB senzor i infracrveni projektor. Ova kamera ima najveće vidno polje u D400 seriji, ali zato ima manju gustoću piksela. Svojom cijenom od 300$ i dokazanim performansama daje vrlo dobar omjer uloženo – dobiveno. Pri odabiru kamere bitno je razmisliti koji način okidanja („shutter type“) kamera posjeduje. Na slici 6 prikazan je izgled, dimenzije i sastav kamere.

Slika 6. Intel RealSense D435 kamera [3]

Neke kamere u D400 seriji imaju takozvani *Rolling shutter record* koji brzo skenira sve piksele scene s lijeva, s desna ili vertikalno, te mu za to treba nekoliko frameova koji se onda spreme u jedan frame. Problem kod ovoga je što nekada ne može kvalitetno slikati objekte koji su u brzom pokretu. Kamera D435 ima takozvani *Total (Global) shutter* koji sve piksele hvata odjednom. Za primjenu u ovome radu dobro je da kamera ima *Total shutter* jer objekti su u pokretu tijekom generiranje fotografija za treniranje neuronske mreže, iako moguće je da bi i *Rolling Shutter* bio dovoljno brz. Kod *Rolling Shutter-a* postoji mogućnost da bi fotografije ispale nešto zamućenije. Oba principa završe obradu slike u isto vrijeme, samo što *Total shutter* uhvati odjednom pa procesuira, dok *Rolling shutter* hvata dio po dio i odmah procesuira. Pri nabavki kamera treba obratiti pozornost da ako će primjena kamera biti hvatanje predmeta u pokretu, kamera koja ima *Total Shutter* je bolja opcija.



Slika 7. Rolling i Total Shutter princip [4]

Ove kamere imaju IR projektor koji projicira infracrvenu svjetlost koja služi za dobivanje dubinske mape i za poboljšanje rezolucije te smanjenje šumova. Koristi se infracrvena svjetlost kako ljudsko oko ne bi vidjelo projekciju. IR projektor pomaže u uvjetima lošijeg svjetlosnog okruženja. Treba naglasiti kako je ovo ipak jeftina kamera, pa i projektor nema dobre performanse.

Još neke karakteristike Intel RealSense D435 kamere:

1. raspon 0.3 -10 m (realnije 0.3 – 3 m)
2. točnost dubine je <2% na udaljenosti od 2 m
3. široko vidno polje: 87° x 58°
4. dubinski frame rate do 90 fps
5. RGB senzor rezolucije 2 MP.

Treba biti svjestan koje su mogućnosti ovih kamera. Prednosti i ograničenja ovih kamera dokazane u realnoj primjeni. Neke od prednosti su:

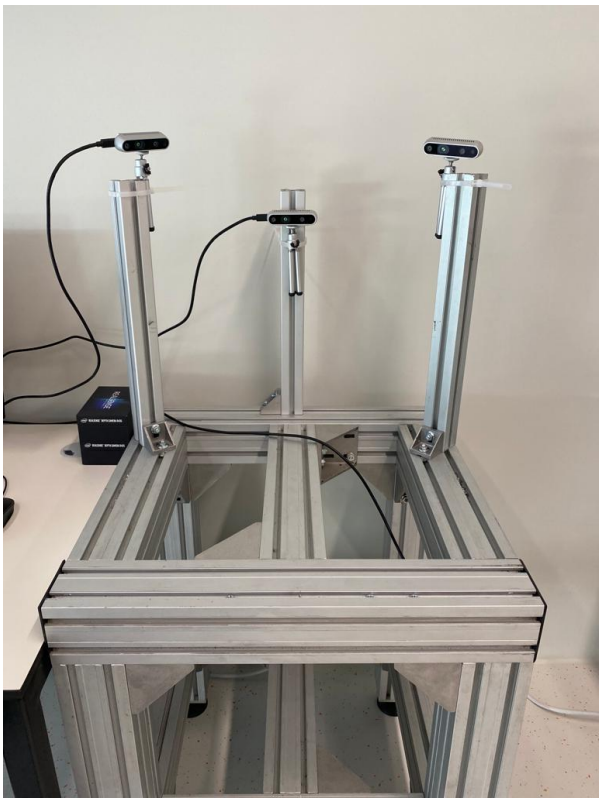
1. radi vani, čak i na najjačoj sunčevoj svjetlosti
2. niska cijena
3. dobra i čvrsta kvaliteta izrade, može izdržati slabije udarce bez degradacije performansi
4. odlična potpora od strane Intela s ažuriranjem drivera
5. radi dobro na svakom većem operativnom sustavu (Linux, Windows, Mac itd.).

Neke od pokazanih ograničenja su:

1. generalno ima velik nivo šuma
2. raspon koji se može koristiti za dalju obradu je manji od 3 m
3. nekada ima problema s hvatanjem manjih geometrijskih detalja.

3. PROJEKTIRANJE POSTAVA I KONSTRUIRANJE NOSAČA ZA KAMERU

Potrebno je da postav s kamerama ima određenu krutost, kako ne bi dolazilo do neželjenog pomicanja kamera. Zbog stereo kalibracije kamera iznimno je bitno da kamere stalno budu u fiksom položaju, zato što ako dođe do pomaka neke od kamera, posljedica će biti netočna stereo kalibracija i potencijalno netočna 3D rekonstrukcija geometrije predmeta. Pomak kamere mogao bi utjecati i na generiranje seta fotografija za treniranje neuronskih mreža jer pomakom kamere mijenja se i svjetlost koja upada na tu kameru, te bi tada filtracija pozadine mogla biti problem. Postav je namontiran pomoću aluminijski profila dimenzija 90 x 90 mm i 45 x 45 mm. Izgled prvog postava prikazan je na slici 8, gdje je vidljivo kako su kamere povezane s postavom s običnim vezicama. To je bilo dovoljno dobro u početnoj fazi diplomskog, dok je bila faza upoznavanja s C++ programskim jezikom i upoznavanje kako upravljati kamerama.



Slika 8. Laboratorijski postav – prva verzija

Nakon početne faze, došlo je vrijeme da se napravi postav sa stalnom konfiguracijom kamera. Odluka je bila da je najbolje napraviti simulaciju postava s konfiguracijom kamera kako bi se

pronašlo optimalno rješenje. Kako su ove kamere relativno jeftine, ima primjena gdje se koristi po tri, četiri, čak i više kamera, primjer na slici 9.

Slika 9. Neke konfiguracije Intel-ovih kamera [5]

Kod izrade ovoga rada odlučili smo se limitirati na tri kamere, iako se moglo koristiti i više, a najmanji mogući broj (za ovaj rad) bio bi dvije kamere. Za početak bilo je potrebno konstruirati nosač za kamere.

3.1. Konstrukcija nosača kamere

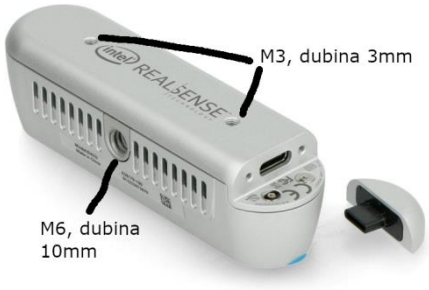
Nosač kamere morao je biti dovoljno robustan da se može čvrsto zategnuti na aluminijski profil i da se kamera može čvrsto zategnuti za njega. Neke debljine stijenki su jednostavno uvjetovane normiranim vijcima koji se inače koriste za pritezanje alu-profila za međusobno.

Slika 10. Vijak za alu-profil

Druge debljine stijenki određene su vijcima po standardu DIN 7991 i dubinom provrta na Intel kamerama. Na slici 11 je prikazan vijak.

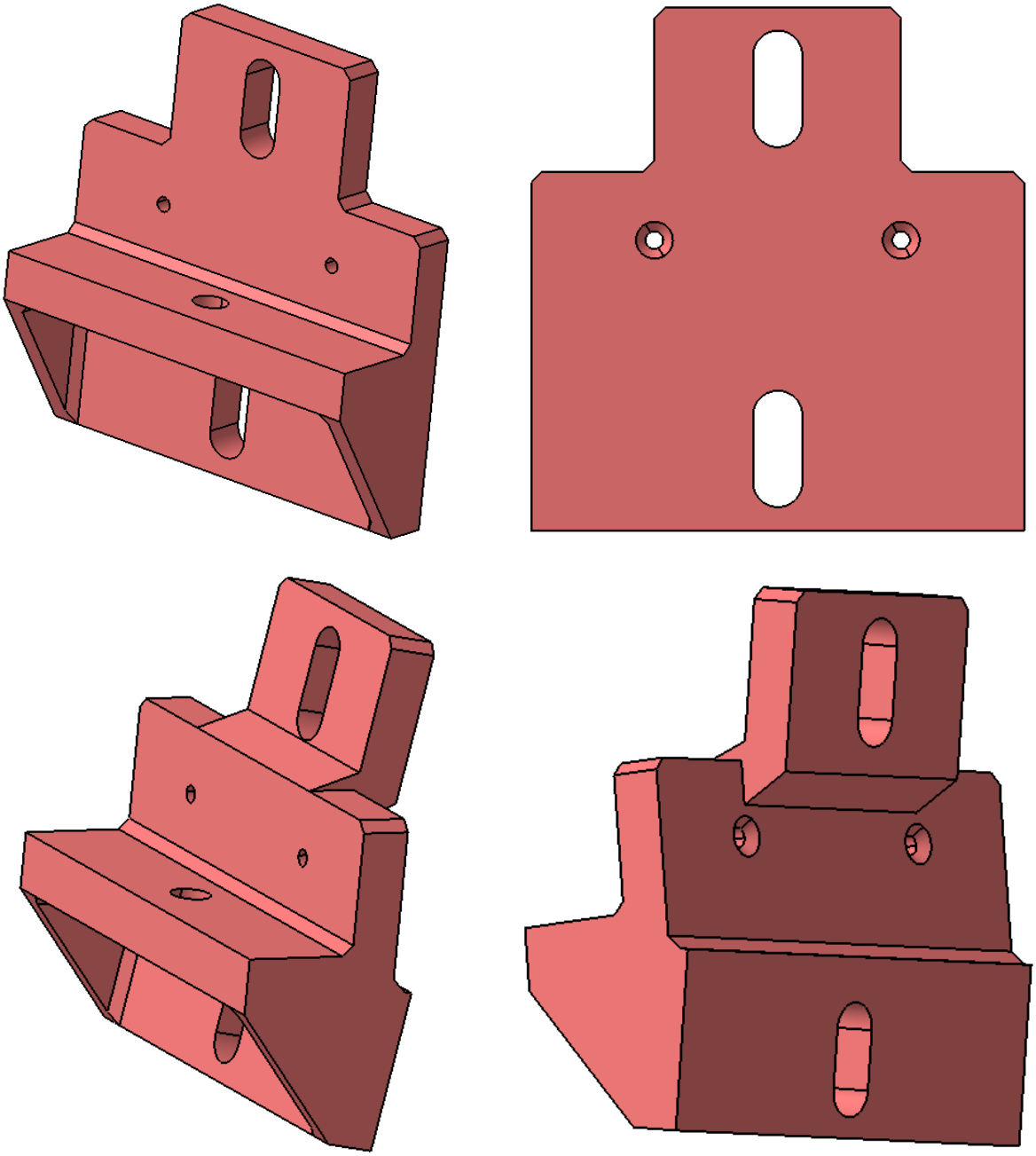
Slika 11. Vijak DIN 7991 M3

Na slici 12 nalazi se kamere s dimenzijama provrta. Dovoljno je kameru pričvrstiti samo na provrt od ispod ili na dva provrta sa zadnje strane.



Slika 12. Kamera s dimenzijama provrta

Izgled nosača kamere prikazan je na slici 13. Izrađen je 3D printanjem i kamere su pričvršćene pomoću dva vijka sa zadnje strane. Srednja kamera montirana je na nosač bez nagiba, dok su druge dvije kamere montirane na nosače s nagibom, razlika između nosača vidi se na slici koja slijedi. Ideja je bila da se dvije kamere sa strane montiraju nešto iznad srednje kamere i nagib im zatim omogućava da vide i gornju plohu predmeta koju srednja kamera ne vidi. Kut nagiba lijevog i desnog nosača iznosi 10°. Postavlja se pitanje zašto nije jedna kamera montirana gore, jedna na sredini i jedna da gleda predmet odozdo. Problem s takvom konfiguracijom bio bi manja konzistentnost kod 3D rekonstrukcije, koja je osjetljiva i mora se paziti na detalje kako bi se kontinuirano ostvarivali dobri rezultati.

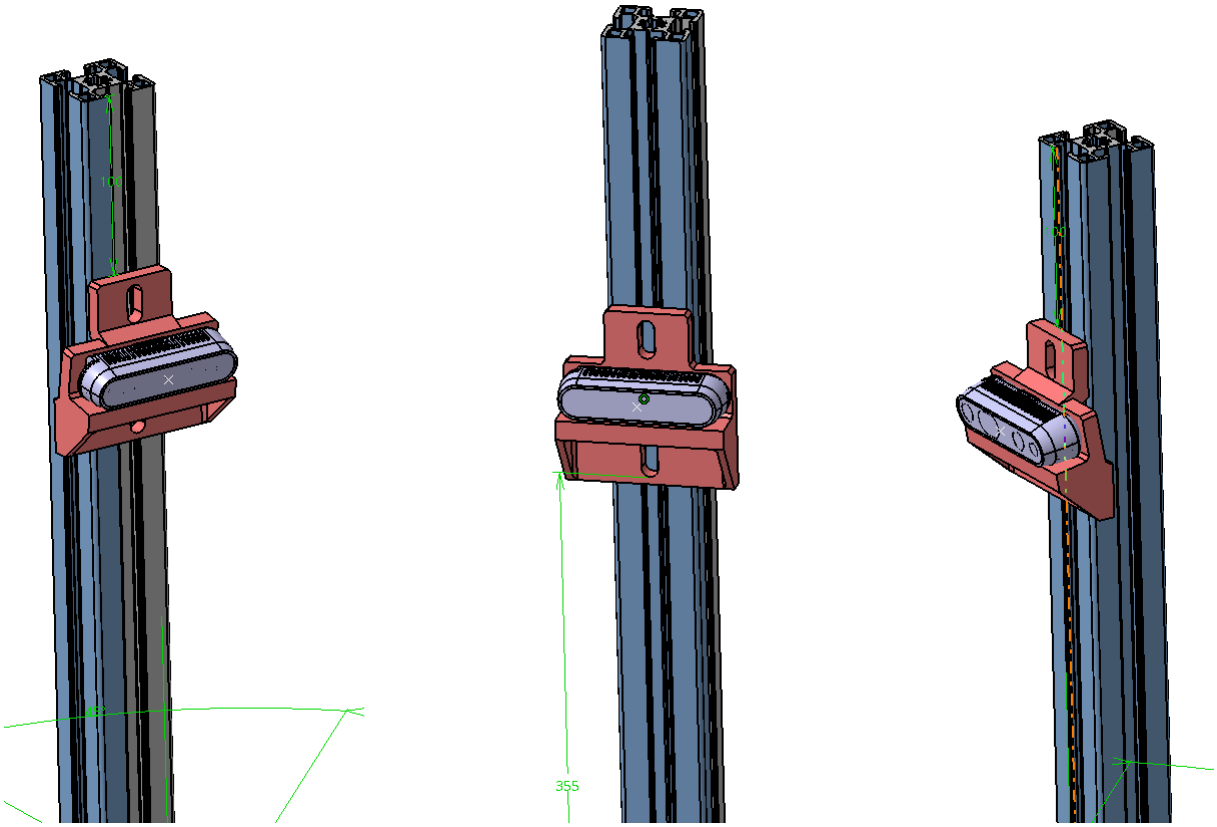


Slika 13. Nosač kamere u CATIA-i

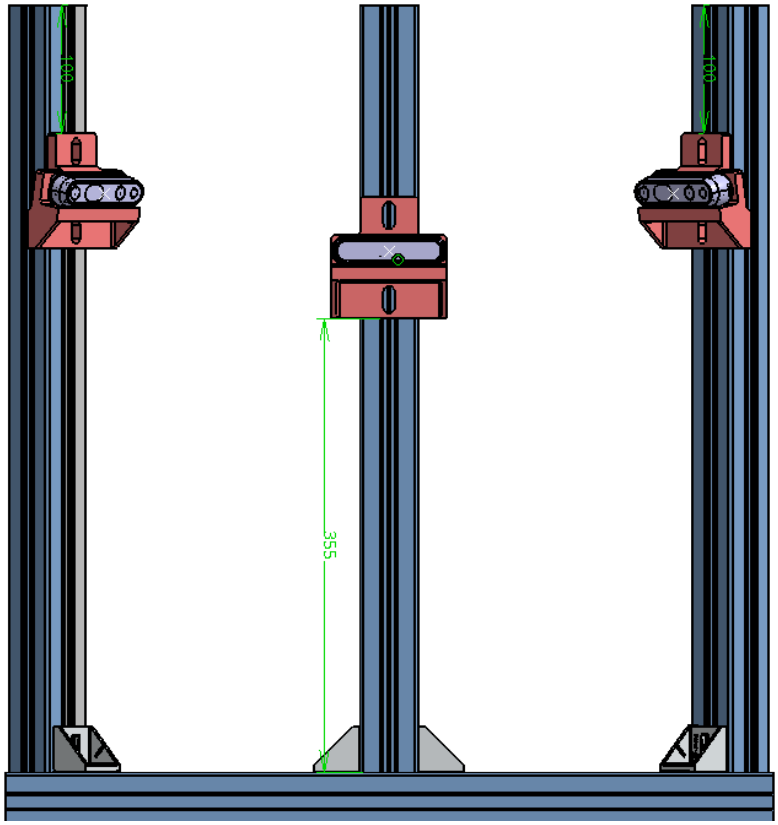
3.2. Simulacija postava

Simulacija postava je napravljena u programskom paketu CATIA V5R20. Razlog izrade simulacije postava može se objasniti na način da bi alternativa bila uzimati slike i oblake točaka na realnom postavu dok god ne bi dobili zadovoljavajuće pozicije kamera. A ako se kasnije želi promijeniti postav ponovno bi se morao ponavljati proces. Zato je simulacija odlična stvar i danas se koristi u mnogim aspektima industrije. Jedino prva izrada simulacije postava traje nešto duže, kasnije izmjene su relativno brze i sigurno brže od trženja idealne

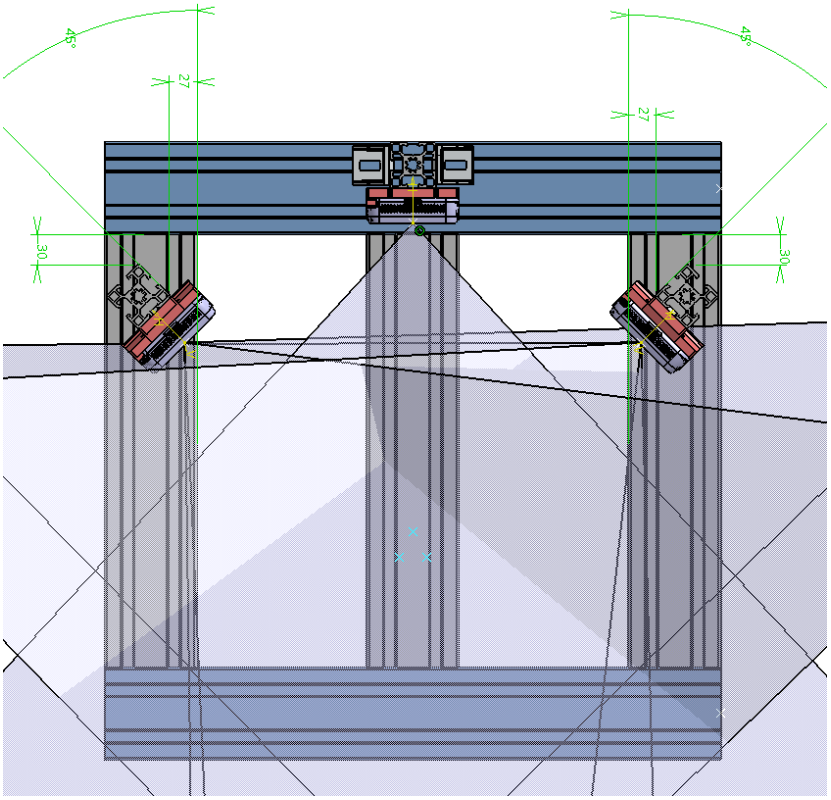
pozicije na pravom postavu. STEP datoteke aluminijskih profila svih dimenzija moguće je pronaći na mnogim mjestima na internetu. STEP datoteka kamere preuzeta je s Intel-ove stranice. Kod pozicioniranja kamera bilo je bitno da ne bude prevelik kut između njih zbog stereo kalibracije. Stereo kalibracija je teoretski moguća do kuta od 90°, ali prema isprobanom s ovim kamerama za dobre rezultate mora biti puno manje od toga. Na sljedeće tri slike prikazana je simulacija postava iz tri pogleda. Na slici 16 se vidi vidno polje ovih kamera koje je vrlo široko, što za ovu primjenu nije optimalno, ali pokazalo se da su rezultati dovoljno dobri da nema potrebe kupovati Intel-ovu kameru iz iste serije s oznakom D415 koja ima duplo manje vidno polje – što znači duplo gušći raspored piksela.



Slika 14. Simulacija postava – izometrijski pogled



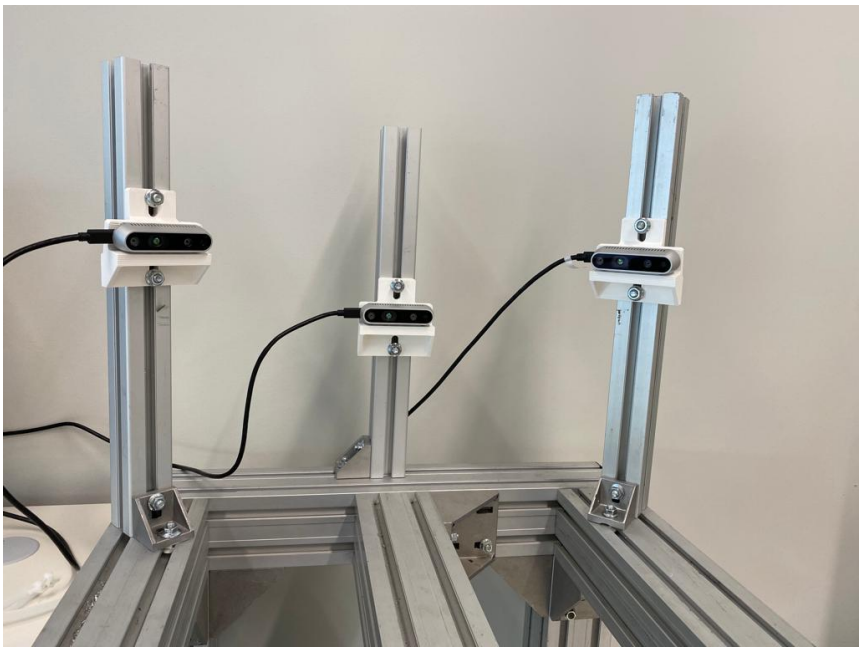
Slika 15. Simulacija postava – pogled od naprijed



Slika 16. Simulacija postava – pogled odozgo

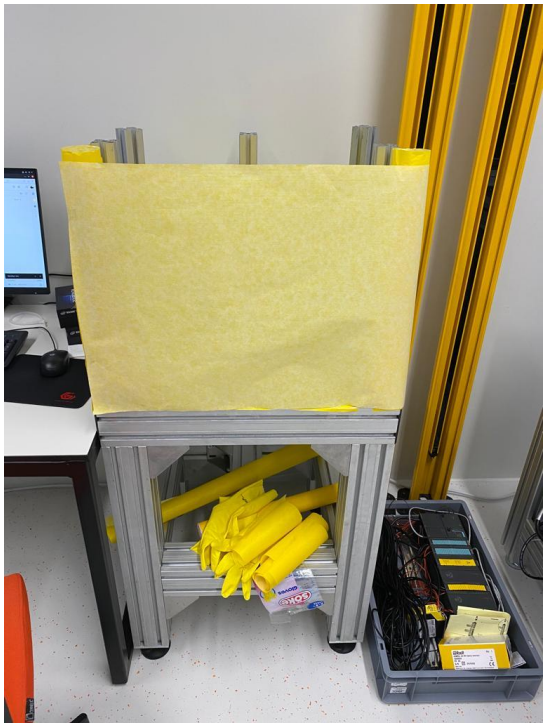
3.3. Razvoj postava u laboratoriju

Na slici 8 bio je prikazan početni postav koji je služio svrsi par mjeseci i zbog čestih slučajnih pomicanja kamera usavršio sam tehniku stereo kalibracije kamera. Kao što je ranije rečeno sljedeći korak bio je montaža kamera na nosače i postavljanje alu profila s kamerama prema pozicijama predviđenim simulacijom.



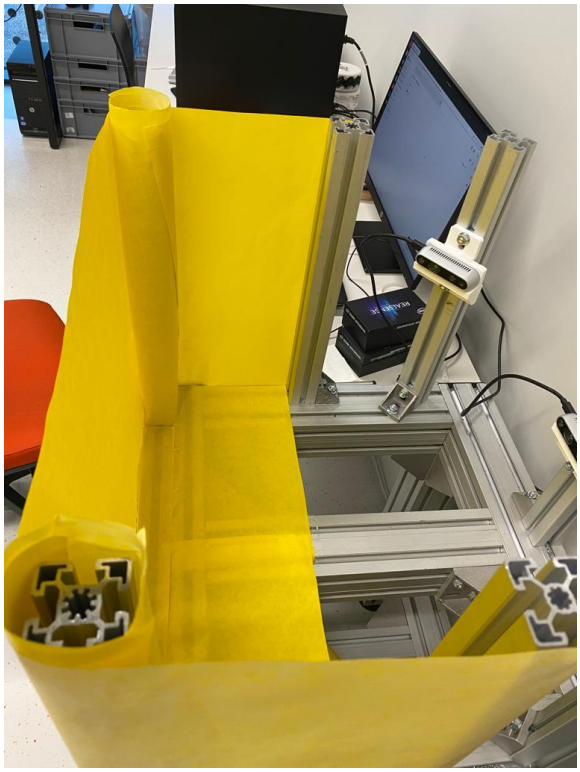
Slika 17. Laboratorijski postav – kamere montirane

Dalje tijek misli vodi u smjeru da je potrebno napraviti pozadinu koju je moguće lako filtrirati iz slike. Najčešće se za ovakve aplikacije koristi fluorescentno zelena ili narančasta boja jer su najrjeđe u proizvodima. Izolirana pozadinama je postignuta tako da se oko nekoliko dodanih alu-profila, na postav postavio papir žute boje – žuta je boja koja će biti segmentirana na slikama. Bitna napomena je da proizvodi koji imaju žutu boju sličnu papiru će isto biti segmentirani, to je nedostatak segmentiranja pozadine. Bez segmentacije pozadine i 3D rekonstrukcija bi bila znatno otežana, a automatsko generiranje slika gotovo nemoguće, zato je za postav pričvršćen papir žute boje. Uz pozadinu od žutog papira, koristile su se i rukavice žute boje za držanje predmeta kod slikanja. Bitno je da rukavice budu što sličnije nijanse kao papir kako bi se morao što uži raspon žute boje segmentirati iz slike. Žute rukavice su vidljive na slici 18 dolje. Sustav se može nadograditi da robot drži željeni proizvod ispred kamera i tada bi se robotska hvataljka morala napraviti da bude slične boje kao i pozadina. Slijede slike konačnog postava nakon postavljene željene konfiguracije kamera i postavljene izolirajuće pozadine.



Slika 18. Laboratorijski postav – prikaz 1

Slika 19. Laboratorijski postav - slika 2



Slika 20. Laboratorijski postav – slika 3

4. INTRINZIČNA KALIBRACIJA KAMERE

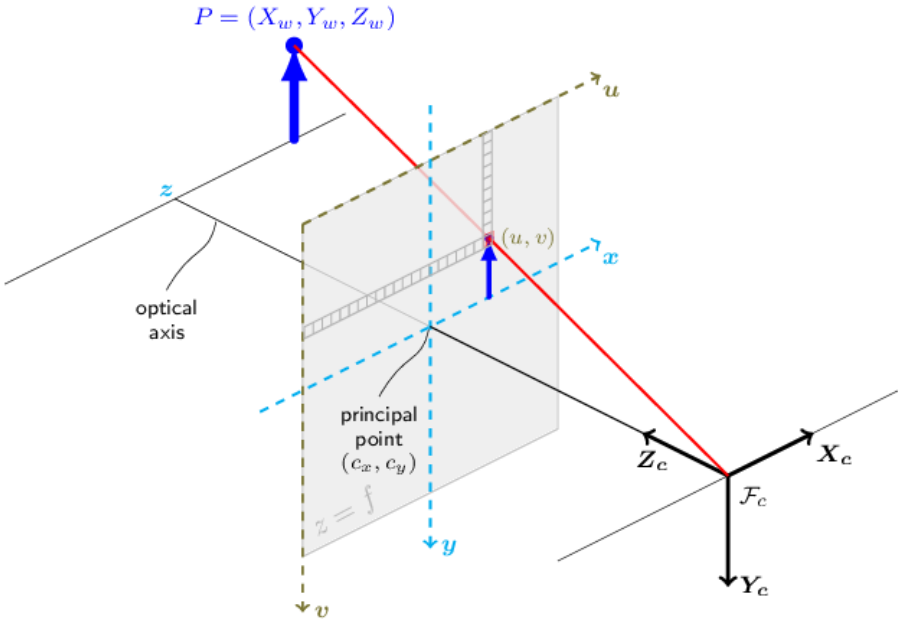
Proces estimacije parametara kamere naziva se kalibracija kamere. To znači da imamo sve informacije kako bi precizno odredili odnos između 3D točke u stvarnom svijetu i odgovarajuće 2D projekcije (piksela) na slici uslikanoj kalibriranom kamerom. Kalibracijom kamere obično se dobivaju intrinzični parametri – žarišna duljina (focal length), optički centar i koeficijenti distorzije leće. Intrinzična matrica pokazuje optička svojstva leće, te njeni parametri služe za ispravljanje distorzije leće, mjerenje objekta i određivanje položaja kamere. Ima više načina na koji se kamera može kalibrirati:

1. Kalibracijskim uzorkom: kada imamo potpunu kontrolu nad procesom uzimanja slika. Najbolji način za izvođenje kalibracije je snimanje nekoliko slika objekta ili uzorka poznatih dimenzija s različitih točaka gledišta. Najčešće su metoda temeljena na šahovnici ili kružnim uzorcima poznatih dimenzija.
2. Geometrijskim uzorcima: ponekad imamo druge geometrijske uzorke kao što su ravne linije ili točke nestajanja koje se mogu koristiti za kalibraciju.
3. Temeljeno na dubokom učenju: kada imamo vrlo malo kontrole nad postavkom slike (npr. imamo jednu sliku scene), još uvijek je moguće dobiti informacije o kalibraciji kamere pomoću metode temeljene na dubokom učenju.

U ovom radu koristit će se kalibracija šahovnicom poznatih dimenzija.

4.1. Geometrija formiranja slike

Kako bi se kamerama i dobivenim slikama mogli izvršavati industrijski zahtjevi, bitno je razumjeti kako kamera projicira 3D prostor na 2D sliku. Za razumijevanje ovog problema bitno je poznavati pozadinsku matematiku. Velik dio formula i slika 21 uzeti su iz [6].
Za primjer se uzima takozvani „pinhole“ model kamere.



Slika 21. Pinhole model kamere [6]

2D slike se dobije projiciranjem 3D točaka **Pw** na ravninu slike koristeći perspektivnu transformaciju koja tvori odgovarajući piksel **p**. Oboje **Pw** i **p** predstavljeni su u homogenim koordinatama, tj. kao 3D i 2D homogeni vektori. Kako bi skratili zapis često se ispusti riječ „homogeni“ i govorimo samo vektor, zna se da se misli na homogeni u ovom slučaju.

Projektivna transformacija bez distorzije koju daje model pinhole kamere prikazana je sljedećim matematičkim zapisom:

$$\mathbf{p} = \mathbf{K}\left[\mathbf{R}|\mathbf{t}\right]\mathbf{P}_w. \qquad \textbf{(6)}$$

Gdje je $\mathbf{P_w}$ je 3D točka izražena u odnosu na globalni koordinatni sustav, **p** je 2D piksel na ravnini slike, **K** je intrinzična matrica kamere, **R** i **t** su rotacija i translacija koji opisuju promjenu s globalnih koordinata na koordinate kamere.

Kao što je ranije navedeno, intrinzična matrica **K** projicira 3D točke dane u koordinatnom sustavu kamere u 2D koordinate piksela:

$$\mathbf{p} = \mathbf{K} \cdot \mathbf{Pc}. \qquad \textbf{(7)}$$

Intrinzična matrica K sastoji se od žarišne duljine f_x i f_y, koje su izražene preko jedinice piksela, i od optičkog centra c_x i c_y koje je obično blizu centra slike:

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}. \qquad \textbf{(8)}$$

Tako da je:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}. \quad \textbf{(9)}$$

Matrica intrinzičnih parametara ne ovisi o sceni koju vidi, tako da, jednom kad se procjeni, može se koristiti dok god je žarišna duljina fiksna, a žarišna duljina je nepromjenjiva u kamerama, jedino eventualno nekim udarcem bi se mogao dogoditi neki pomak unutar kamere.

Rotacijsko-translacijska matrica $\left[\mathbf{R}|\mathbf{t}\right]$ povezuje globalne koordinate s koordinatama kamere te je matrični produkt projektivne i homogene transformacije. Projektivna transformacija 3x4 preslikava 3D točke predstavljene u koordinatama kamere u 2D točke ravninu slike i prezentirane su u normaliziranim koordinatama kamere $x' = X_c / Z_c$ i $y' = Y_c / Z_c$:

$$Z_c \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix}. \quad \textbf{(10)}$$

Homogena transformacija kodirana je ekstrinzičnim parametrima **R** i **t** i predstavlja promjenu baze iz globalnog koordinatnog systema *w* u koordinatni sustav kamere *c*. Dakle, s obzirom na prikaz točke P u globalnim koordinatama, *Pw*, dobivamo prikaz točke P u koordinatnom sustavu kamere *Pc*, pomoću:

$$P_c = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} P_w. \quad \textbf{(11)}$$

Ova homogena transformacija sastoji se od matrice rotacije **R**, dimenzija 3x3, i vektora translacije **t**, dimenzija 3x1:

$$\begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \textbf{(12)}$$

stoga slijedi:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}. \quad \textbf{(13)}$$

Kombinirajući projektivnu transformaciju i homogenu transformaciju, dobivamo projektivnu transformaciju koja preslikava 3D točke iz globalnih koordinata u 2D ravninu slike u normaliziranim koordinatama kamere:

$$Z_c \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} R|t \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}. \quad \textbf{(14)}$$

Gdje su $x' = X_c / Z_c$ i $y' = Y_c / Z_c$. Ako stavimo intrinzične i ekstrinzične jednadžbe zajedno možemo $\mathbf{p} = \mathbf{K}\begin{bmatrix} \mathbf{R}|\mathbf{t} \end{bmatrix}\mathbf{P}_w$ zapisati kao:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}. \quad \textbf{(15)}$$

Ako je $Z_c \neq 0$, prošla transformacija jednaka je sljedećoj jednadžbi:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} f_x X_c / Z_c + c_x \\ f_y Y_c / Z_c + c_y \end{bmatrix}$$

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} \mathbf{R}|\mathbf{t} \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}. \quad \textbf{(16)}$$

Realne leće obično imaju određenu distorziju, više radijalnu i manje tangencijalnu distorziju. Zato model koji smo dosad pokazali možemo proširiti kao:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} f_x x'' + c_x \\ f_y y'' + c_y \end{bmatrix}, \quad \textbf{(17)}$$

gdje su:

$$\begin{bmatrix} x'' \\ y'' \end{bmatrix} = \begin{bmatrix} x' \dfrac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + 2 p_1 x' y' + p_2 (r^2 + 2x'^2) + s_1 r^2 + s_2 r^4 \\ y' \dfrac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + p_1 (r^2 + 2y'^2) + 2 p_2 x' y' + s_3 r^2 + s_4 r^4 \end{bmatrix}, \quad \textbf{(18)}$$

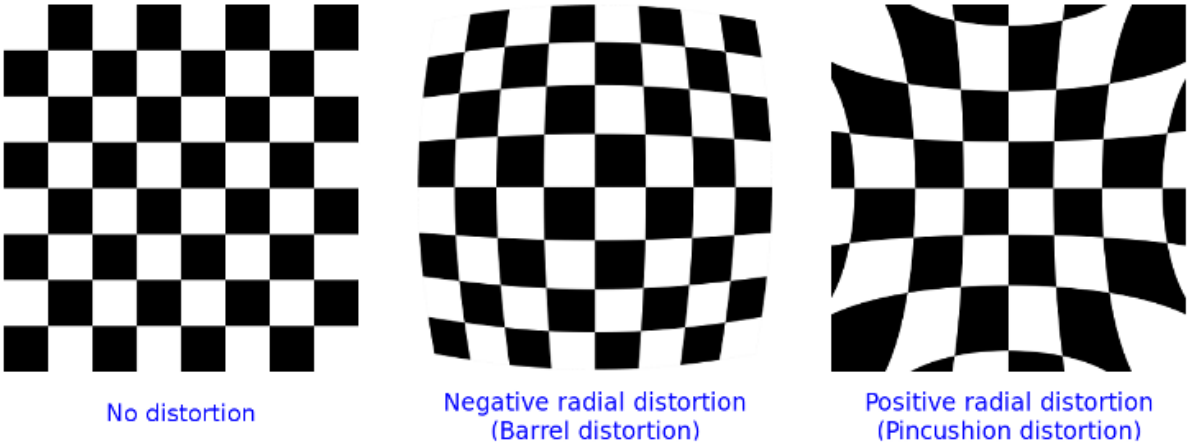
gdje su:

$$r^2 = x'^2 + y'^2, \tag{19}$$

i:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} X_c / Z_c \\ Y_c / Z_c \end{bmatrix}. \tag{20}$$

Distorzijski parametri su radijalni koeficijenti k_1, k_2, k_3, k_4, k_5 i k_6. Tangencijalni distorzijski koeficijenti su p_1 i p_2. I s_1, s_2, s_3, s_4 su koeficijenti distorzije tanke prizme. Koeficijenti višeg reda se ne uzimaju u obzir u OpenCV. Slika 22 prikazuje oblike distorzije.



Slika 22. Oblici distorzije [6]

4.1.1. Homogene koordinate

Homogene koordinate su sustav koordinata koji se koristi u projektivnoj geometriji. Njihova uporaba omogućuje predstavljanje beskonačnih točaka konačnim koordinatama i pojednostavljuje formule u usporedbi s kartezijskim koordinatama, npr. imaju prednost da se „affine“ transformacija može izraziti kao linearna homogena transformacija.

Homogeni vektor $\mathbf{P_h}$ dobiva se dodavanjem jedinice na kraj n-dimenzijskog kartezijskog vektora $\mathbf{P}$, npr. za 3D kartezijski vektor preslikavanje $\mathbf{P} \rightarrow \mathbf{P_h}$:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}. \tag{21}$$

Za inverzno mapiranje vektor $\mathbf{P_h} \rightarrow \mathbf{P}$, podijele se svi elementi homogenog vektora s njegovim zadnjim elementom, npr. ako iz 3D homogenog vektora želimo dobiti 2D kartezijski:

$$\begin{bmatrix} X \\ Y \\ W \end{bmatrix} = \begin{bmatrix} X / Z \\ Y / Z \end{bmatrix}. \quad \textbf{(22)}$$

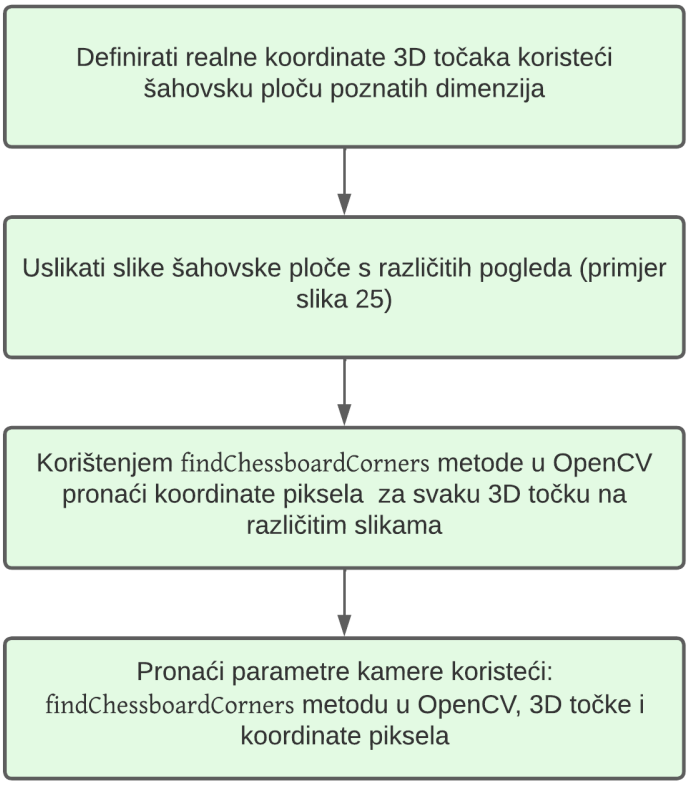
Kao što je ranije spomenuto, korištenjem homogenih koordinata možemo izraziti bilo koju promjenu baze parametrizirane s **R** i **t** kao linearnom transformacijom, npr. za promjenu baze iz koordinatnog sustava 0 u koordinatni sustav 1:

$$\mathbf{P_1} = \mathbf{R} \cdot \mathbf{P_0} + \mathbf{t} \rightarrow \mathbf{P_{h_1}} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \mathbf{P_{h_0}}. \quad \textbf{(23)}$$

4.2. Kalibracija kamere pomoću OpenCV

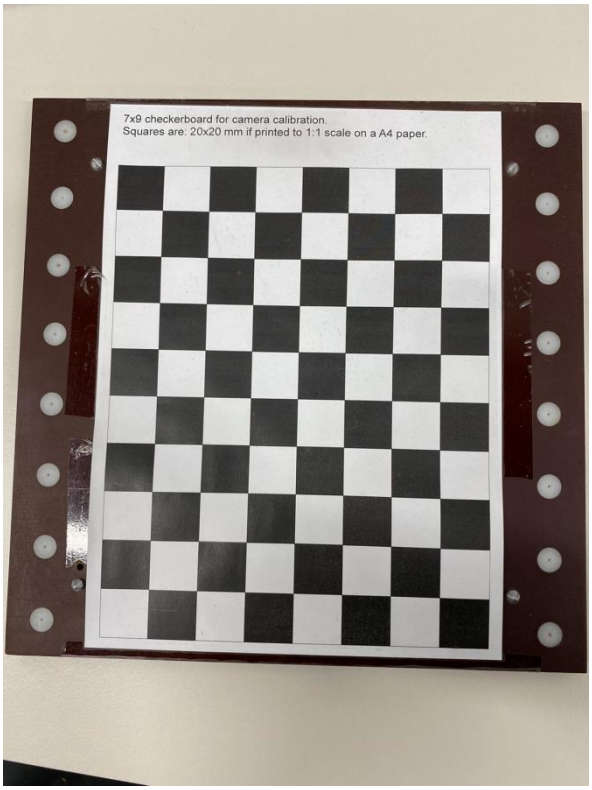
Kako je ranije spomenuto kalibracija kamere je proces procjene intrinzičnih parametara te parametara distorzije leće.

Kalibracijski proces može se prikazati sljedećim dijagramom toka:



Slika 23. Dijagram toka intrinzične kalibracije kamere

Ranije je spomenuto kako se za kalibraciju najčešće koristi šahovska ploča ili kružnice. Svaki odabir ima svoje prednosti i mane, te izbor ovisi o kamerama i primjeni. U ovom projektu kalibracija se radi šahovskom pločom prikazanom na slici 24.



Slika 24. Šahovska ploča za kalibraciju kamere

Bitno je znati dimenzije šahovske ploče – veličinu i strukturu, te je bitno imati printer dovoljne preciznosti, jer svako odstupanje kod šahovske ploče će imati utjecaj na kvalitetu kalibracijskog postupka kamere, dobiti će se manje točne intrinzična matrica i matrica koeficijenata, te će svaki dalji rad od uzimanja slika, do obrade slika biti lošije kvalitete. Dimenzije ove šahovske ploče označene su na njoj i iznose: 7x9 presjeka kvadratića i s dimenzijom pojedinog kvadratića od 20 mm.

4.2.1. Proces uzimanja slika za intrinzičnu kalibraciju kamere

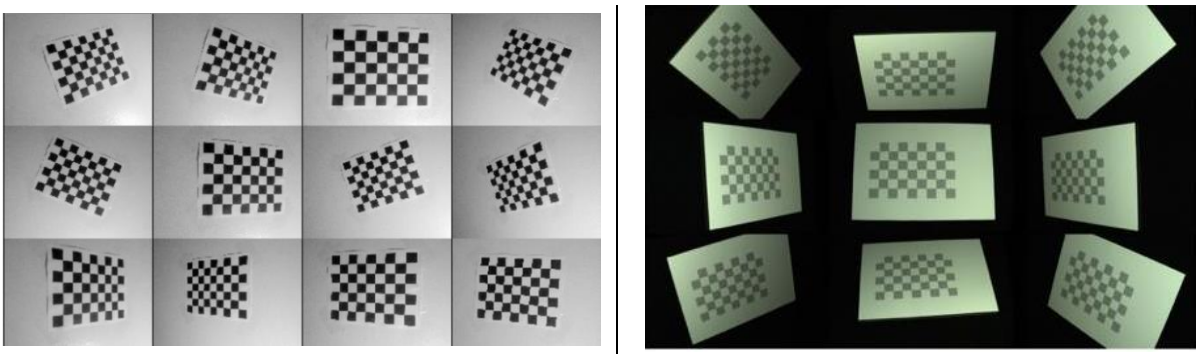
Najveća pitanja su ovdje kakve slike uzeti i koliko slika uzeti?

Savjeti na internetu jako variraju, od 5-100 slika. Odgovor je da se ne može dati generalan odgovor jer on ovisi o programu u kojemu se vrši kalibracija, nemaju svi programu isti algoritam kalibracije kamere. Zato je najbolje isprobati što se više mogućnosti može. U MATLABU se preporučuje 10-20 slika, a rezultati su pokazali da je tako i u OpenCV-u.

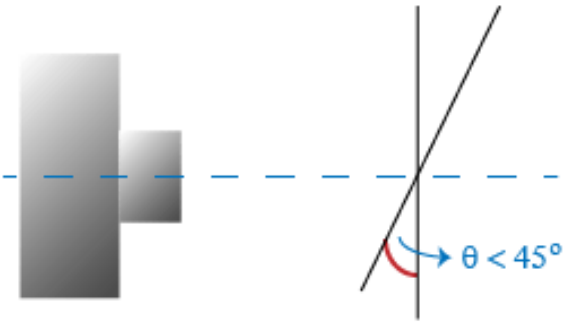
Od velike je važnosti kako/koje slike uzeti?

Isprobano je puno načina uzimanja slika te je na kraju zaključak: optimalno je 9 ili 16 slika uzetih na način kao što je prikazano na slici 25. Bitno je ići redom iz npr. gornjeg lijevog kuta i proći gornji red, pa sve srednje slike, te na kraju zadnji red slika. Bitno je staviti ploču pod

kuteve kako je na slici ispod prikazano, ali ne ići dalje od kuta prikazanog na slici 26. Ne treba se vraćati dvaput na isto mjesto gdje je slika već uzeta, najčešće to povećava grešku.

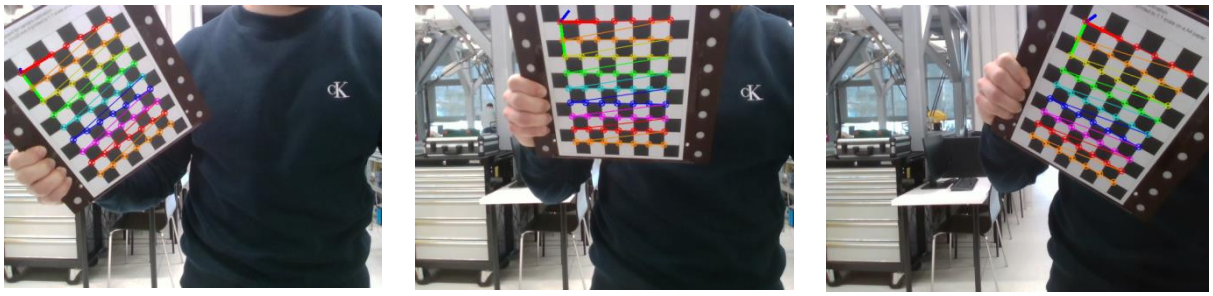


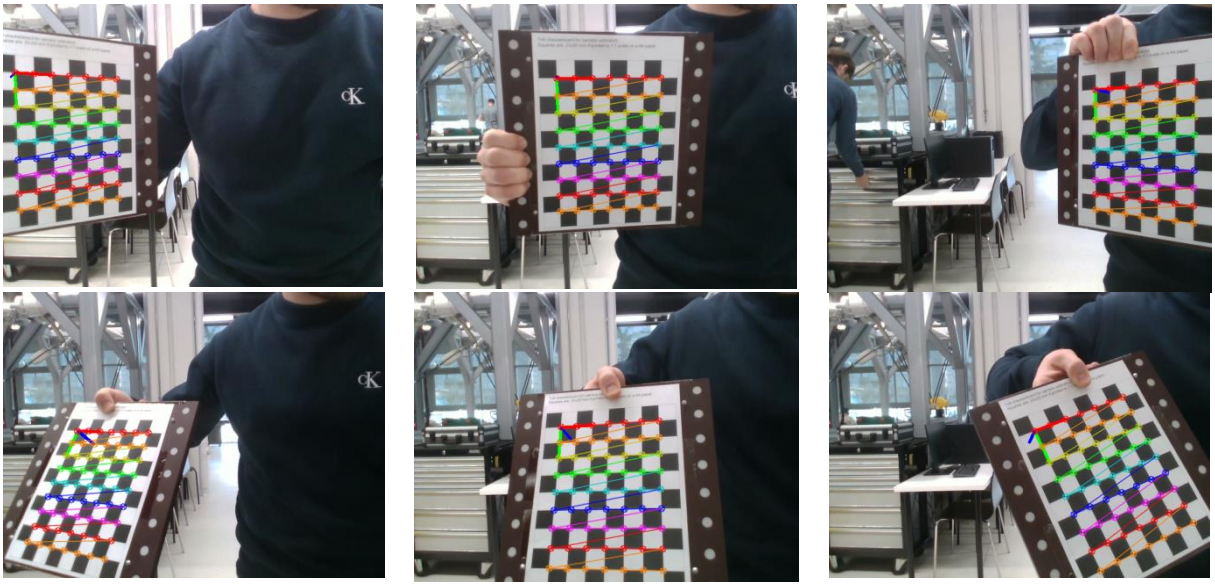
Slika 25. Primjer uzimanja fotografija za kalibraciju



Slika 26. Ograničenje kuta kod uzimanja fotografija za kalibraciju [7]

Primjer slika uzetih za kalibraciju jedne kamere Intel RealSense D435 prikazane su na slici 27. Vidljivo je kako je šahovska ploča prošla sve pozicije u objektivu kamere. Kod kalibracije ove kamere treba se paziti na udaljenost od kamere. U specifikacijama se može pronaći da je idealna udaljenost 0.3-3 m. Postoje preporuke da se kalibracija (i stereo kalibracije) rade na udaljenosti na kojoj će se kasnije nalaziti i predmet tijekom rada, jer bi onda na toj udaljenosti kamera mogla biti najtočnija.





Slika 27. Fotografije uzete za kalibraciju jedne kamere

Kada su fotografije uzete kreće se s kalibracijom. Sljedeći kod pokazuje funkciju za kalibraciju kamere *calibrateCamera()* u OpenCV-u, u kojoj su prva tri argumenta ulazna, a ostala četiri izlazna. Za izlazne argumente dobiju se intrinzična matrica i matrica distorzije, te procijenjeni vektori rotacije i translacije za svaki uzorak. Bitan je *rms* koji je tipa *double* te nam daje procijenjenu grešku kalibracije, cilj je da on bude što manji. Vrijednost *rms*<1 smatra se u redu, a optimalno je *rms* <0.5. Funkcija je vidljiva na dnu sljedećeg *snippeta*. Prikazane su sve deklaracije varijable, jer u C++ sve mora biti egzaktno definirano, to je jedan od dvostrukih mačeva C++ programskog jezika. *idStr1* varijabla sadrži lokaciju i ime slike te omogućava da se pomoću *for* petlje svih 9 slika učita kada je potrebno. Slike se učitavaju jedna po jedna i prolaze potrebne procese kako bi se dobile sve informacije potrebne za kalibraciju kamere.

```
Mat img, imgs, gray;
vector<Point2f> corners;
vector<vector<Point2f>> imagePoints;
vector<vector<Point3f>> objectPoints;
vector<Mat> rvecs, tvecs;
Mat intrinsic = Mat(3, 3, CV_32FC1);
Mat distCoeffs;
double rms;
intrinsic.ptr<float>(0)[0] = 1;
intrinsic.ptr<float>(1)[1] = 1;

for (int i = 0; i < 9; i++)
{
    string idStr1 = "images_saved_cam" + to_string(nrCam) + "/pic" + to_string(i+1) + ".png";
    img.empty();
    img = imread(idStr1);
    cvtColor(img, gray, COLOR_BGR2GRAY);
    bool found1 = false;
    found1 = findChessboardCorners(img, patternSize, corners,
                                   CALIB_CB_ADAPTIVE_THRESH | CALIB_CB_FILTER_QUADS);

    if (found1)
    {
        cornerSubPix(gray, corners, Size(5, 5), Size(-1, -1),
                     TermCriteria(CV_TERMCRIT_EPS | CV_TERMCRIT_ITER, 30, 0.1));
        drawChessboardCorners(gray, patternSize, corners, found1);
        //            imshow(idStr1, gray1);

        vector< Point3f > obj;
        for (int i = 0; i < boardHeight; i++)
        {
            for (int j = 0; j < boardWidth; j++)
            {
                obj.push_back(Point3f((float)j * squareSize, (float)i * squareSize, 0));
            }
        }
        imagePoints.push_back(corners);  // imagePoints
        objectPoints.push_back(obj);   // objectPoints
    }
    else
    {
        cout << "Chessboard corners not found!" << endl << endl;
    }
}

rms = calibrateCamera(objectPoints, imagePoints, img.size(), intrinsic,
                      distCoeffs, rvecs, tvecs, CALIB_FIX_K4|CALIB_FIX_K5);
```

Slika 28. Snippet za kalibraciju kamere

Redom će biti objašnjeno značenje ulaznih argumenata i kako doći do njih.

- **objectPoints –** je vektor vektora točaka uzorka kalibracije u koordinatnom sustavu uzorka kalibracije. Vanjski vektor sadrži onoliko elemenata koliko je broj prikaza uzorka (u slučaju sa slike 9. to će biti vektor koji sadrži 9 vektora).

Kako bi se ovaj vektor vektora ispunio točkama, najlakše je pomoću *for* petlje i potrebno je znati dimenzije ploče, koje su ranije bile navedene. Ova for petlja zapravo prolazi po cijeloj ploči s lijeva na desno i odozgor prema dolje. Koliko slika ima toliko puta će se ova for petlja postupak u petlji ponoviti.

- **imagePoints** – vektor vektora projekcija točaka uzorka kalibracije.

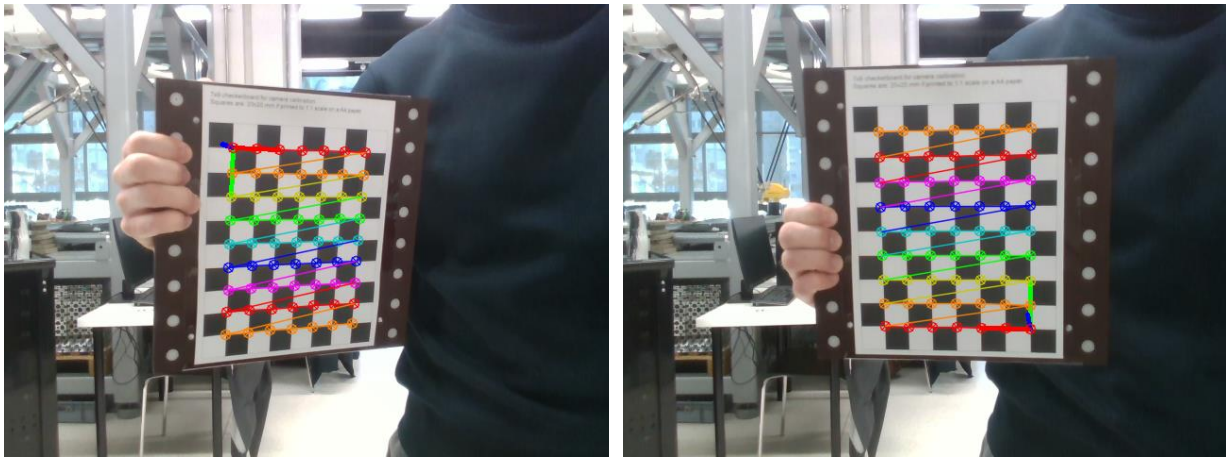
Do točaka projekcija dolazi se pomoću OpenCV funkcije *findChessboardCorners*.

U funkciju idu ulazni argumenti *image* `i` *patternSize*, prvo je slika šahovnice koju smo fotografirali, te mora biti 8-bitna crno-bijela ili slika u boji. A pattern size je ranije bio naveden, to je dimenzija šahove ploče. Izlazni argument je *corners* što će biti vektor koji sadrži detektirane kutove.

Na kraju su takozvane zastavice, prva će konvertirati sliku u crno-bijelu kako bi se smanjio utjecaj svjetlosti na detekciju kutova. A druga poboljšava robusnost. Za svaku sliku ćemo pronaći kuteve, te zatim svaki puta samo ih spremati redom kako dođu u varijablu *imagePoints*.

- **img.size()** – dimenzija slike, svaka fotografija će imati iste dimenzije, tu se samo uzme dimenzija bilo koje.

Napomena: kod uzimanja fotografija koje će se koristiti za kalibraciju kamere, najbolje je odmah pri uzimanja slike tražiti pomoću OpenCV funkcija da se nacrtaju pronađeni kutevi i koordinatna os na uzetoj slici. To je jedini način da budemo sigurno kako su kutevi dobro pronađeni i os jednako okrenuta na svim fotografijama, te da slika nije zamućena. Kao što je prikazano na slici 29. Na lijevoj slici je koordinatna os u lijevom gornjem kutu, dok je na desnoj u desnom donjem kutu. Ukoliko bi samo jedna slika imala drugačije okrenutu os od ostalih, krajnji rezultat bila bi netočna kalibracija.



Slika 29. Prikaz pravilno i nepravilno pronađene koordinatne osi na fotografiji

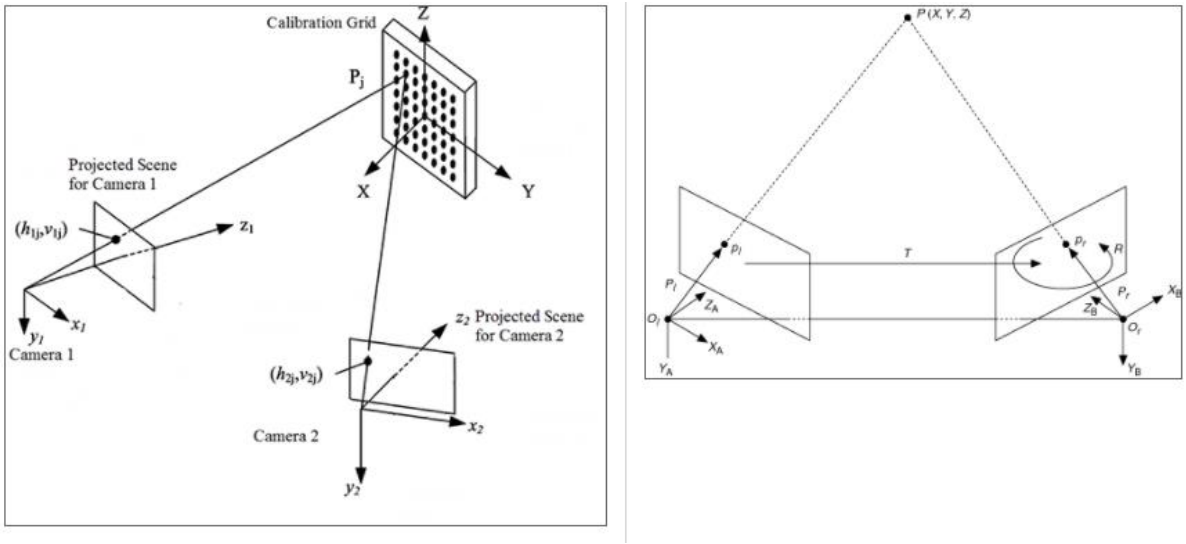
Funkcije kojima se iscrtavaju kutevi i koordinatne osi su *drawFrameAxes()* i *drawChessboardCorners()*. U obje funkcije prvi argument je ulazno-izlazni, to je fotografija na koju želimo da se rezultat iscrta. Drugi argument u *drawChessboardCorners()* je ulazni, to su dimenzije ploče. Zadnja dva argumenta se dobiju iz funkcije *findChessboardCorners()* koja je objašnjena ranije. U funkciji *drawFrameAxes()* ostali argumenti su ulazni, s lijeva na desno, intrinzična matrica, matrica distorzije, vektor rotacije zatim vektor translacije koji zajedno dovode točke iz koordinatnog sustava modela u koordinatni sustav kamere, te na kraju dimenzija kvadrata na šahovskoj ploči za kalibraciju. Na sljedećem *snippetu* prikazana je funkcija napravljena u C++ koja se poziva svaki put kada se žele iscrtati koordinatne osi i pronađeni kutevi. Na dnu snipetta se vide OpenCV funkcije *drawChessboardCorners()* i *drawFrameAxes()*.

```
void fDrawAxisAndCorners(Mat imgDraw, vector<Point2f> cornersDraw, vector< Point3f > objDraw, string izlazDraw, bool foundDraw)
{
    vector<Point2f> objectPointsPlanar;
    for (size_t i = 0; i < objDraw.size(); i++)
    {
        objectPointsPlanar.push_back(Point2f(objDraw[i].x, objDraw[i].y));
    }

    vector<Point2f> imagePoints;
    undistortPoints(cornersDraw, imagePoints, cameraMatrix1, distortionCoefficients1);

    Mat H = findHomography(objectPointsPlanar, imagePoints);

    double norm = sqrt(H.at<double>(0,0)*H.at<double>(0,0) +
                       H.at<double>(1,0)*H.at<double>(1,0) +
                       H.at<double>(2,0)*H.at<double>(2,0));
    H /= norm;
    Mat c1  = H.col(0);
    Mat c2  = H.col(1);
    Mat c3 = c1.cross(c2);
    Mat tvec = H.col(2);
    Mat R1(3, 3, CV_64F);
    for (int i = 0; i < 3; i++)
    {
        R1.at<double>(i,0) = c1.at<double>(i,0);
        R1.at<double>(i,1) = c2.at<double>(i,0);
        R1.at<double>(i,2) = c3.at<double>(i,0);
    }
    //cout << "R (before polar decomposition):\n" << R << "\ndet(R): " << determinant(R) << endl;
    Mat W, U, Vt;
    SVDecomp(R1, W, U, Vt);
    R1 = U*Vt;
    //    cout << "R (after polar decomposition):\n" << R1 << "\ndet(R): " << determinant(R1) << endl;
    Mat rvec;
    Rodrigues(R1, rvec);
    drawFrameAxes(imgDraw, cameraMatrix1, distortionCoefficients1, rvec, tvec, 2*squareSize);

    drawChessboardCorners(imgDraw, patternSize, cornersDraw, foundDraw);
    imshow(izlazDraw, imgDraw);
    waitKey(1);
}
```

Slika 30. Snippet iscrtavanje koordinatne osi i pronađenih kuteva na fotografiji

5. EKTRINZIČNA (STEREO) KALIBRACIJA KAMERE

Za stereo kalibraciju kamera (kao i za kalibraciju jedne kamere) postoji nekoliko mogućnosti: Matlab, ROS, OpenCV itd. Svaka od opcija ima svoje prednosti i mane, ali više imaju sličnosti nego razlika. Ovdje će biti prikazana provedba stereo kalibracije pomoću postojeće funkcije u OpenCV *stereoCalibrate()*. Odabran je princip da se prvo 9 parova slika uslika i svakoj provjeri valjanost pomoću iscrtavanja koordinatne osi i pronađenih kutova. Nakon što se spreme slike koje zadovoljavaju, kreće primjena algoritama kako bi dobili potrebne ulazne argumente za funkciju *stereoCalibrate()*. Uglavnom je postupak isti kao kod kalibracije jedne kamere, potrebni su *objectPoints* i *imagePoints*, samo što su ovdje 2 argumenta *imagePoints* jer su dvije kamere te će svaka imati svoje *imagePoints*. Prema njima se i pronalazi stereo kalibracija. Sustav stereo kamera prikazan je na slici 31.



Slika 31. Teorija stereo kalibracije kamera [8]

Funkcija stereo kalibracije u OpenCV je:

```
double rmsStereo = stereoCalibrate(objectPoints, imagePoints[0], imagePoints[1],
                    intrinsic2, distortionCoeff2, intrinsic1, distortionCoeff1, img1.size(),
                        R, T, E, F, CALIB_FIX_INTRINSIC | CALIB_FIX_FOCAL_LENGTH | CALIB_FIX_PRINCIPAL_POINT);
```

Slika 32. Snippet sintaksa stereo kalibracijske funkcije

Ovako se dobije rotacija i translacija prve kamere u odnosu na drugu, a ako se želi dobiti odnos koordinata druge kamere na prvu samo se zamjeni redoslijed *imagePoints* i ostalih parametara vezanih za svaku pojedinu kameru.

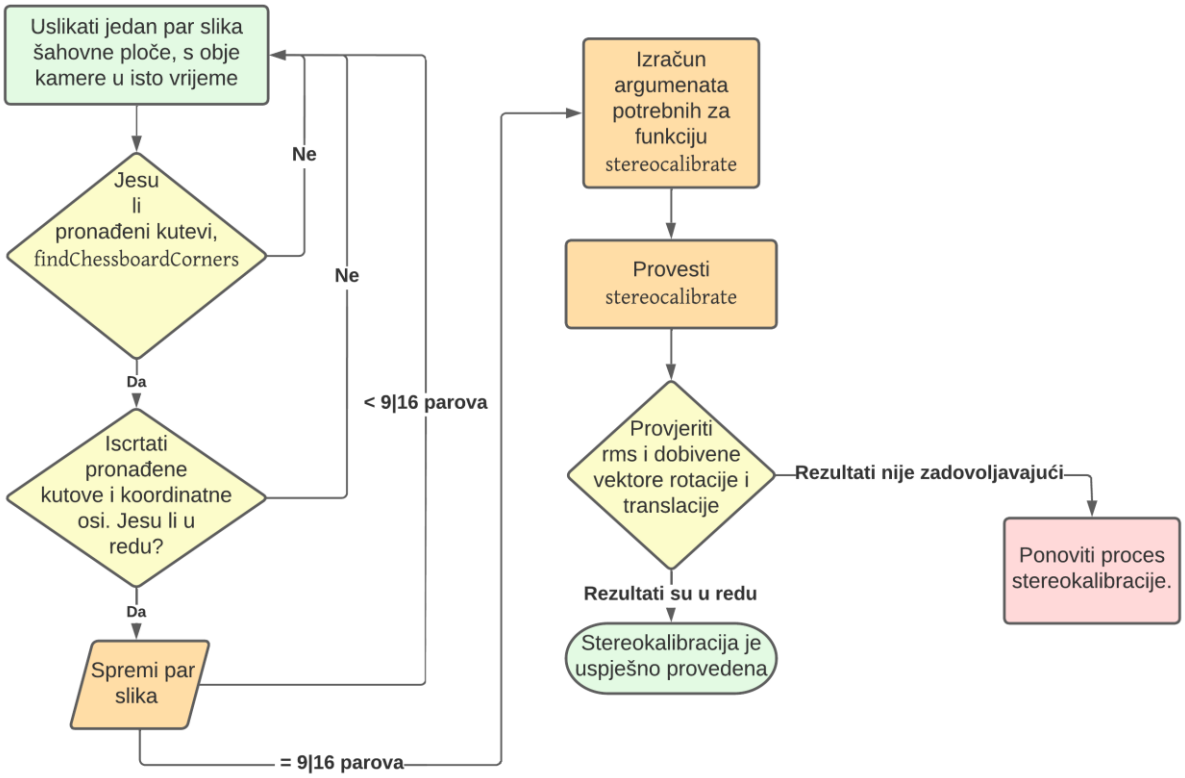
Od ulaznih podataka potrebne su još intrinzične matrice i matrice distorzije za obje kamere, te dimenzija slike. Na kraju funkcije ponovo su zastavice. One mogu imati veći utjecaj na rezultat, zato je važno isprobati više kombinacija zastavica te vidjeti koje daju optimalne rezultate za zadani slučaj koji ovisi o kamerama, primjeni, svjetlosti, predmetima rada itd.

Funkcija ima 4 izlazna argumenta:

- **R** – matrica izlazne rotacije, zajedno s translacijskim vektorom T, pokazuje transformaciju baze koordinatnog sustava prve kamere u točke koordinatnog sustava druge kamere
- **T** – translacijski vektor
- **E** – esencijalna matrica (ne koristi se u ovom projektu)
- **F** – fundamentalna matrica (ne koristi se u ovom projektu).

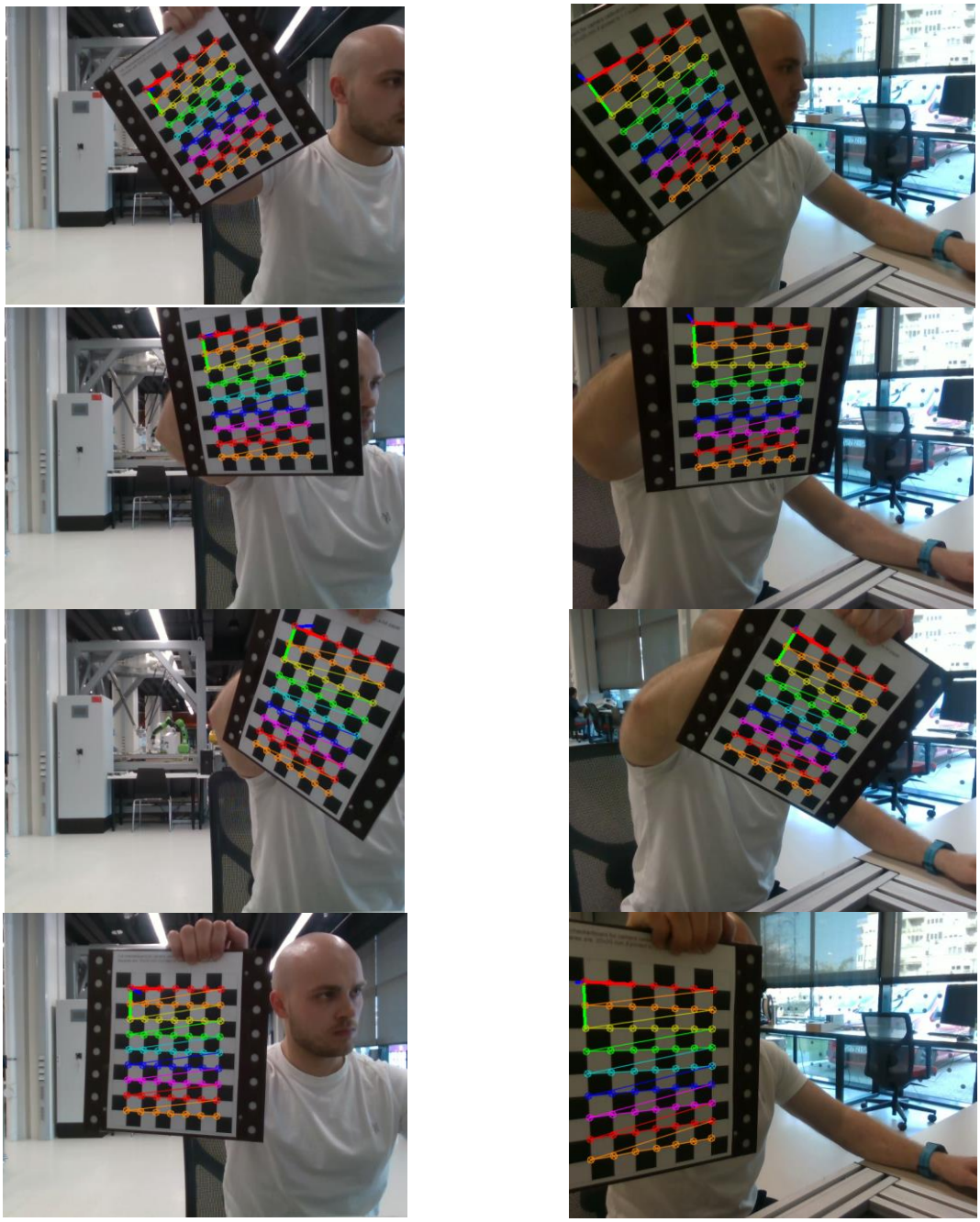
Matrice R i T su bitne za ovaj zadatak jer će se transformirati koordinatni sustav druge kamere u odnosu na prvu te koordinatni sustav treće kamere u odnosu na prvu. Tako će se dobiti tri stereo kalibrirane kamere.

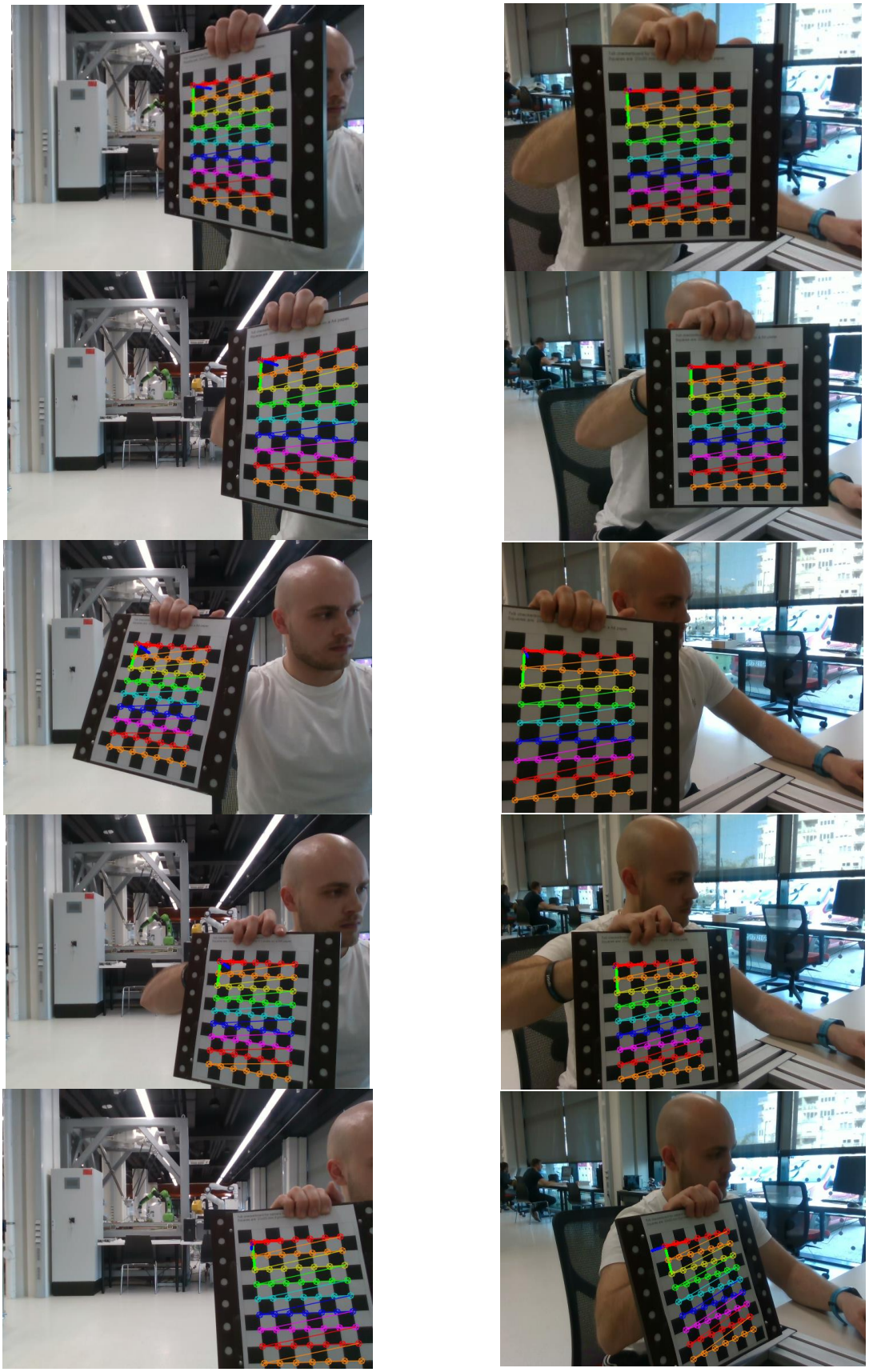
Dijagram toka stereo kalibracije prikazan je na slici 33.



Slika 33. Dijagram toka stereo kalibracije

Primjer slika korištenih u stereo kalibraciji, prema slici 25. desni primjer, prikazan je sljedećim slikama. Može se primijetiti isti tijek uzimanja devet slika po cijelom vidnom polju kamere kao kod kalibracije jedne kamere. Također se da primijetiti kako su slike s desne kamere nešto tamnije, što možda ima utjecaj u onekoj mjeri na točnost stereo kalibracije.





Slika 34. Primjer slika uzetih za stereo kalibraciju

Kada se kamere učvrste za postav dovoljno je jednom uzeti slike s dviju kamera, i zatim funkcijom stereo kalibracije dobiju se rotacija i translacija prve na drugu kameru, te isto tako jednostavnim zamjenom mjesta varijabli u funkciji, mogu se dobiti rotacija i translacija druge na prvu. Kako je ranije spomenuto rezultat stereo kalibracije je matrica transformacije, koja je sadržana od matrica rotacije i translacije i zadnji red je dodan kako bi matrica bila kvadratna. Na sljedećim slikama prikazan je rezultat stereo kalibracije ispisan u terminalu. Ispisani su redom intrinzični podaci prve kamere, podaci druge kamere, greška kalibracije `RMS`, zatim matrica rotacije R, vektor translacije T i na kraju kompletna matrica transformacija.

```
Koju kameru(uz prvu) želite kalibrirati? [2 ili 3]:
2

Podaci 1. kamere:
Intrinsic: [606.4589397155881, 0, 323.8192914516035;
 0, 608.9568521454468, 232.953287308522;
 0, 0, 1]
Distorzija: [0.118528124344197, -0.08440031024789287, 0.001894827182756823, 0.0009566420064575788, -0.5469761647135536]
RMS: 0.154832

Podaci 2.kamere:
Intrinsic: [605.9155623357842, 0, 327.0374950989066;
 0, 608.6992912781304, 249.1025756541105;
 0, 0, 1]
Distorzija: [0.1139140410807251, -0.09305291279895701, 0.0011429685223075, -0.001264114911056868, -0.4709592268953043]
RMS: 0.138515

RMS stereo kalibracije iznosi: 0.216666

Matrica R[0.7434759903704703, -0.2629728368995472, 0.6148892085535834;
 0.09495305124295092, 0.9516373709113477, 0.2921818480751182;
 -0.661987439308558, -0.1588445823373968, 0.7324896100694918]
Matrica T[-227.498, -108.847, 115.835]

Matrica transformacije Cloudova 1-2:
 0.743476 -0.262973  0.614889 -0.227498
0.0949531  0.951637  0.292182 -0.108847
-0.661987 -0.158845   0.73249  0.115835
        0         0         0         1
```

Slika 35. Podaci stereo kalibracije kamera 1-2

```
Koju kameru(uz prvu) želite kalibrirati? [2 ili 3]:
3
Podaci 1. kamere:
Intrinsic: [606.4589397155881, 0, 323.8192914516035;
 0, 608.9568521454468, 232.953287308522;
 0, 0, 1]
Distorzija: [0.118528124344197, -0.08440031024789287, 0.001894827182756823, 0.0009566420064575788, -0.5469761647135536]
RMS: 0.154832

Podaci 3.kamere:
Intrinsic: [607.8706685620882, 0, 326.1014619218584;
 0, 610.6908816737536, 252.8382244044736;
 0, 0, 1]
Distorzija: [0.1216912111964207, -0.1154164345290365, 0.000332558248094442, 0.0005656859927972691, -0.4640211470734437]
RMS: 0.136651

RMS stereo kalibracije iznosi: 0.187996

Matrica R[0.856438802677117, 0.1961251615718545, -0.4775431899497679;
 -0.05995354123848121, 0.9565523080237541, 0.2853300806213651;
 0.512755448690652, -0.2157373472833179, 0.8309869113440989]
Matrica T[238.754, -114.837, 65.8909]

Matrica transformacije Cloudova 1-3:
  0.856439   0.196125  -0.477543   0.238754
-0.0599535   0.956552    0.28533  -0.114837
  0.512755  -0.215737   0.830987  0.0658909
         0          0          0          1
```

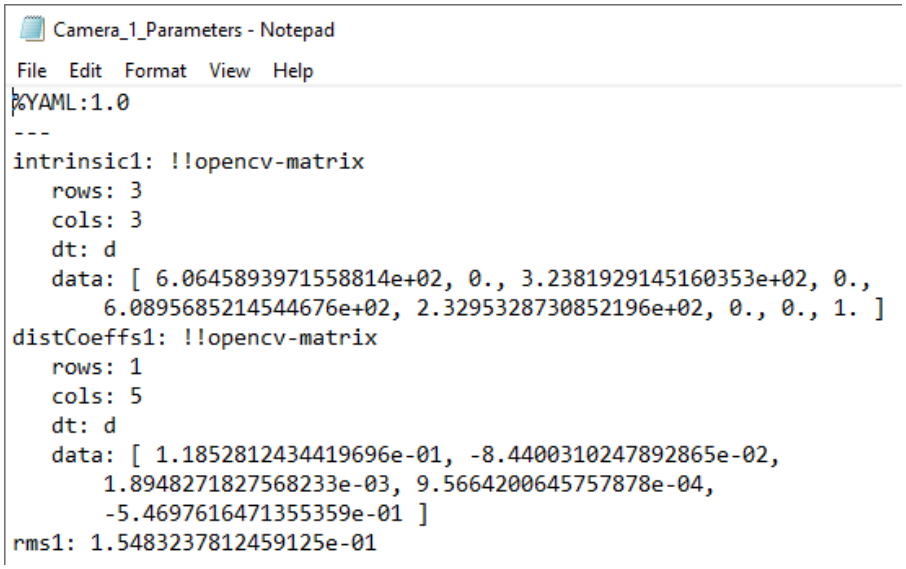
Slika 36. Podaci stereo kalibracije kamera 1-3

5.1. Spremanje dobivenih transformacija

Kalibracija pojedinačne kamere i stereo kalibracija kamera su procesi koji troše dosta resursa i traju određeno vrijeme, a u industriji su često i vrijeme i resursi ograničeni i očekuje se maksimalna efikasnost – sa što manje resursa dobiti što je više moguće. Iz navedenih razloga i uz uvjet da su kamere čvrsto vezane na svoje pozicije, najbolje je jednom napraviti intrinzične kalibracije i ekstrinzične kalibracije te ih spremiti u tekstualnu datoteku. U ovom radu kalibracije su spremane u obliku .yaml datoteke. Korištena je funkcija *FileStorage()* iz knjižnice OpenCV, funkcija omogućava lako i brzo pisanje i čitanje podataka u XML/YAML/JSON tipove podataka. Kod svakog pokretanja programa, u procesu inicijalizacije programa učitavaju se spremljeni podaci o kalibracijama i dodjeljuju im se varijable. Slijedi primjer pisanja rezultata dobivenih intrinzičnom kalibracijom prve kamere i rezultata dobivenih stereo kalibracijom između prve i druge kamere:

```
cv::FileStorage storage("Camera_1_Parameters.yml", cv::FileStorage::WRITE);
storage << "intrinsic1" << intrinsic;
storage << "distCoeffs1" << distCoeffs;
storage << "rms1" << rms;
storage.release();

cv::FileStorage storage("/home/ubuntu-z2/RobustVision_New/Cam2-1_transform.yml", cv::FileStorage::WRITE);
storage << "transformMatrix2-1" << transformMatsave;
storage.release();.
```

Slika 37. Snippet spremanja podataka u .yaml datoteku



Slika 38. Primjer YAML zapisa intrinzičnih parametara kamere

Cam1-2_transform - Notepad

File Edit Format View Help

```
%YAML:1.0
---
transformMatrix1-2: !!opencv-matrix
   rows: 4
   cols: 4
   dt: f
   data: [ 7.20039904e-01, 1.96265802e-02, -6.93655014e-01,
       2.61786997e-01, -1.10655956e-01, 9.90056574e-01, -8.68519396e-02,
       7.14929104e-02, 6.85053110e-01, 1.39293924e-01, 7.15052068e-01,
       6.47487193e-02, 0, 0, 0,
       1. ]
```

Slika 39. Primjer YAML zapisa ekstrinzičnih parametara kamera jedan i dva

Slijedi primjer čitanja spremljenih rezultata.

```
cv::FileStorage storage("/home/ubuntu-z2/RobustVision_New/Camera_1_Parameters.yml", cv::FileStorage::READ);
storage["intrinsic1"] >> intrinsic1;
storage["distCoeffs1"] >> distortionCoeff1;
storage["rms1"] >> rms1;
storage.release();

cv::FileStorage storage("/home/ubuntu-z2/RobustVision_New/Cam2-1_transform.yml", cv::FileStorage::READ);
storage["transformMatrix2-1"] >> matrica1;
storage.release();.
```

Slika 40. Snippet čitanja podataka spremljenih u .yaml datoteku

Ako dođe do pomaka neke kamere, ili se čisti leća kamere – mora se ponovno provesti intrinzična i ekstrinzična kalibracija te kamere.

6. REKONSTRUKCIJA 3D GEOMETRIJE PROIZVODA

Manipulacija 3D podacima znatno je zahtjevnija od 2D podataka. OpenCV knjižnica je godinama u primjeni, u stalnom razvoju, ima dobro napravljenu dokumentaciju, te su vjerojatno najbolja open-source knjižnica za obradu 2D podataka. Na drugu stranu, što se tiče ponude open-source knjižnica za 3D podatke, u ovom radu izbor je bio između PCL (Point Cloud Library) i Open3D.

Prednosti i mane PCL-a:

1. ima puno funkcija
2. relativno je stara knjižnica pa na webu ima puno primjera što se sve radilo
3. sintaksa je kompliciranа
4. nedostatak dobre dokumentacije
5. python verzija PCL-a nije potpuna.

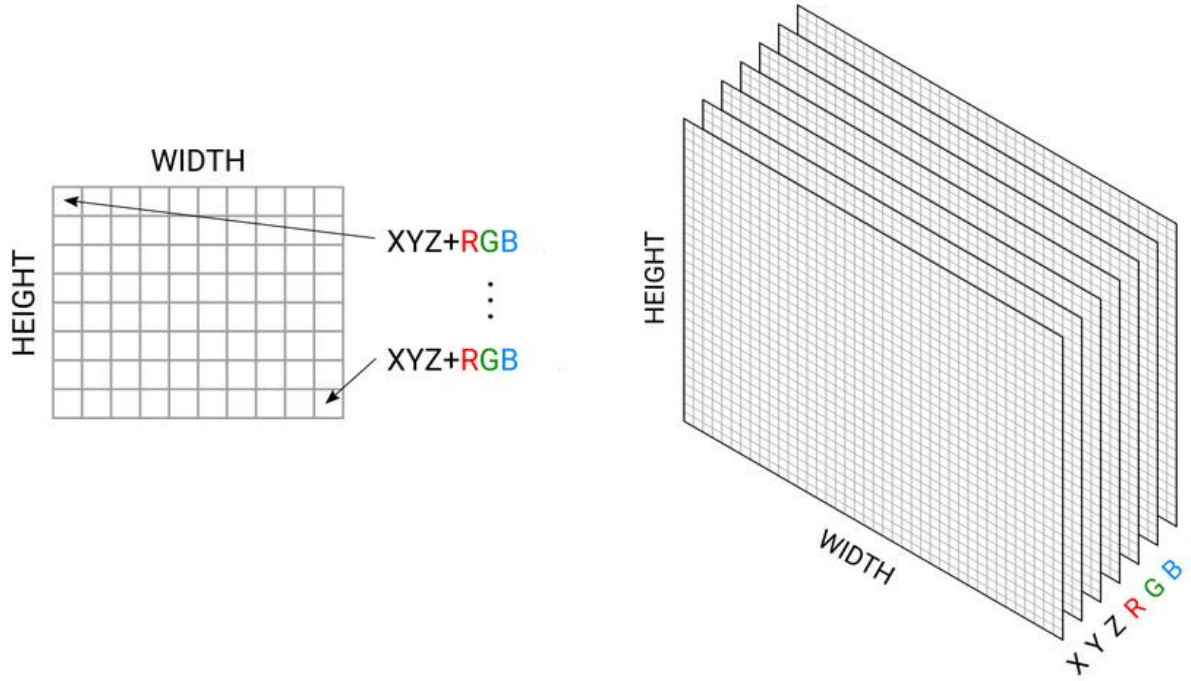
Prednosti i mane Open3D:

1. moderna, nova knjižnica s jednostavnom sintaksom
2. brzo se razvija
3. broj korisnika brzo raste
4. postoji C++ i python potpuna verzija
5. još nema sve funkcije koje npr. PCL
6. Dokumentacija je odlična.

U ovom radu je na početku bila korištena PCL knjižnica, a nakon nekog vremena je sve prebačeno na Open3D knjižnicu u C++ programskom jeziku. Nažalost, dokumentacija Open3D-a je odlična za primjere napravljene u python kodu, dok za C++ je dokumentacija bez primjera, tako da puno za funkcija se moralo gledati u python primjeru i onda uz C++ dokumentaciju shvatiti kako točno funkcija radi. Čak i ovako Open3D je znatno jednostavniji za korištenje od PCL-a zbog sintakse. Svaka funkcija se izvrši u manje redova i sve je intuitivnije. Podatak koliko se koja knjižnica spominje u znanstvenim člancima dobro je mjerilo koja je knjižnica u prednosti trenutno. Taj podatak je da spominjanje PCL-a je u silaznog putanji dok je Open3D sve češće spominjan. PCL je nastao 2010.g, dok je Open3D nastao 2015.g.

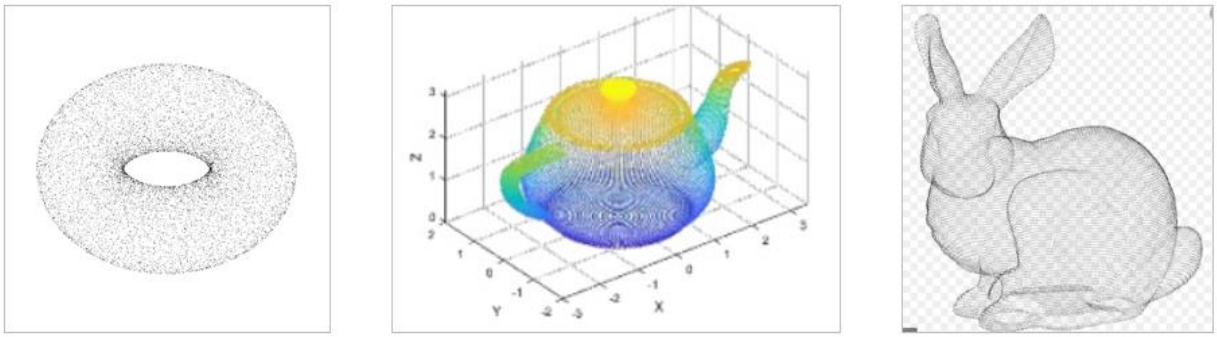
6.1. Oblak točaka (eng. Point Cloud)

Oblak točaka je struktura podataka koja se koristi za predstavljanje skupa točaka u prostoru. Oblaci točaka koriste se u mnoge svrhe, kao što su stvaranje 3D CAD modela, za mjeriteljstvo i inspekciju, i za mnoge druge aplikacije poput vizualizacije, animaciju, renderiranje itd. U ovom radu se oblaci točaka koriste za vizualizaciju i rekonstrukciju geometrije proizvoda. Struktura point clouda je slična kao kod 2D slika, samo uz još dodanu Z koordinatu. Primjer strukture oblaka točaka prikazan je na slici 41.



Slika 41. Struktura oblaka točaka [9]

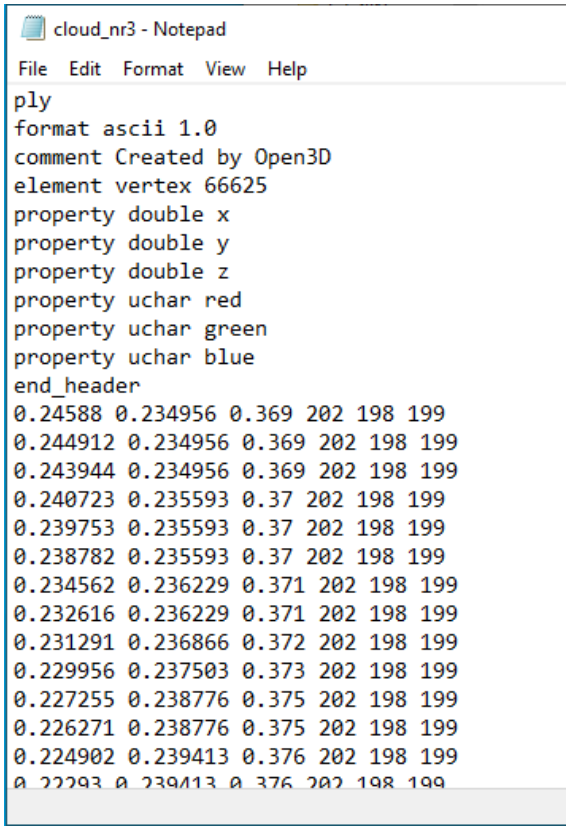
Svaka točka ima svoju x, y i z koordinatu i, ako je oblak točaka u boji, ima još i r, g, b vrijednost. Naravno da su moguće i druge strukture kao: x,y,z i intenzitet, ili x,y,z i rgb vrijednost spremljena u jedan podatak tipa *integer* itd.



Slika 42. Primjer oblaka točaka [10]

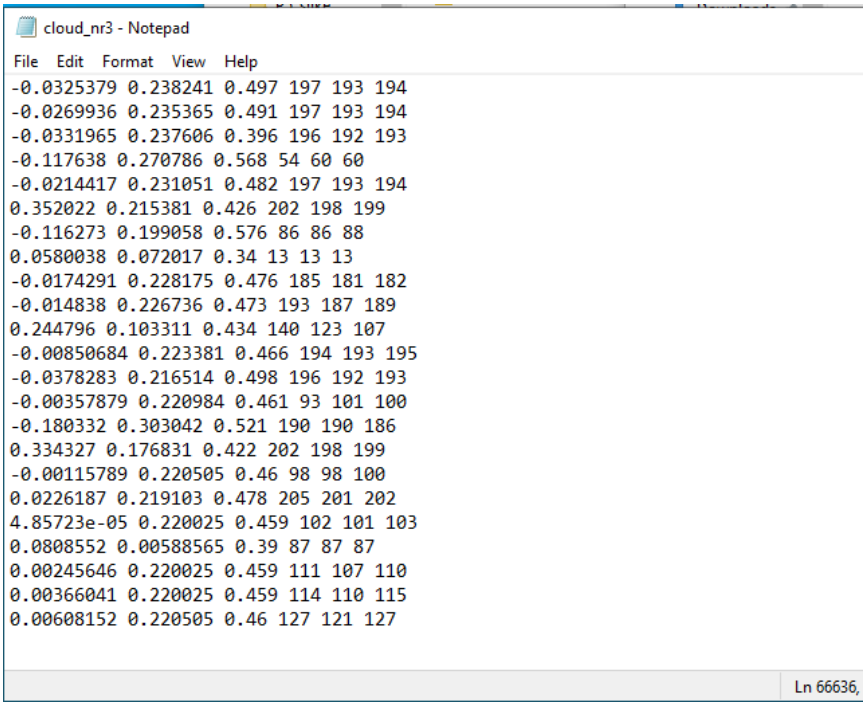
6.2. Veličina oblaka točaka

Veličina oblaka točaka je tema od velike važnosti. Oblak točaka se najčešće sprema u .ply formatu, tako se radilo i u ovome radu. PLY je kratica od Polygon File Format i ovaj format se koristi za pohranu i čitanje 3D podataka sa raznih skenera, kamera itd. Moguće je pohraniti boju, transparentnost, normale na površinu, teksture koordinata itd. Moguće je koristiti ASCII zapis ili binarni format, u ovome radu se koristi ASCII. Ovako pohranjeni oblak točaka može se prikazati brojčanim vrijednostima u programima kao Microsoft Word ili Notepad, primjer iz Notepad-a je prikazan na slici ispod gdje je vidljivo o čemu je ranije pisano: struktura oblaka točaka (x,y,z,r,g,b), .PLY i ASCII format.



Slika 43. Izgled pohranjenog oblaka točaka

'Problem' je u veličini oblaka točaka. Uzmemo li jedan frame s Intelovom kamerom, odnosno jednu sliku i jedan oblak točaka, slika će zauzima 56 KB memorije dok će oblak točaka biti oko 4 MB. Ovo je dobar pokazatelj koliko je 3D svijet kompliciraniji od 2D, barem što se tiče manipulacije i procesuiranja na računalu. Pitanje je kako može oblak točaka biti oko 4 MB, gdje je toliko informacija? Odgovor leži na dnu Notepad-a otvorenog oblaka točaka i prikazan je na slici 44. Desno dolje piše da zapis ovog oblaka točaka sadrži oko 66000 linija x, y, z, r, g, b.



Slika 44. Veličina oblaka točaka

Ova veličina dolazi do izražaja kada se na oblak točaka želi primijeniti neka funkcija. Na primjer u ovom radu se puno koristi ICP o kojem će kasnije biti detaljnije objašnjenje, ali ta funkcija koja zahtjeva kao ulazne argumente dva oblaka točaka, gotovo da se ne može izvršiti ako se u nju stave dva oblaka točaka od 4 MB jer funkcija zahtjeva puno resursa. Onda je opcija koristiti pomoć grafičke kartice, što je danas vrlo poznato s programom kao CUDA ili OpenCL. A druga opcija je smanjiti veličinu oblaka točaka, naravno, uz uvjet da kvaliteta oblaka točaka ostane gotovo nepromijenjena.

6.2.1. Smanjenje uzorkovanja (Voxel downsampling)

Odličan alat za ovakav problem dan je u Open3D knjižnici u vidu funkcije VoxelDownSample(). Funkcija ima jedan ulazni argument, to je veličina voksela koja se unosi u metrima. Sintaksa funkcije je prikazana ispod.

```
cloudPtr = point_cloud.VoxelDownSample(0.002);
```

Slika 45. Snippet sintaksa *VoxelDownSample()*

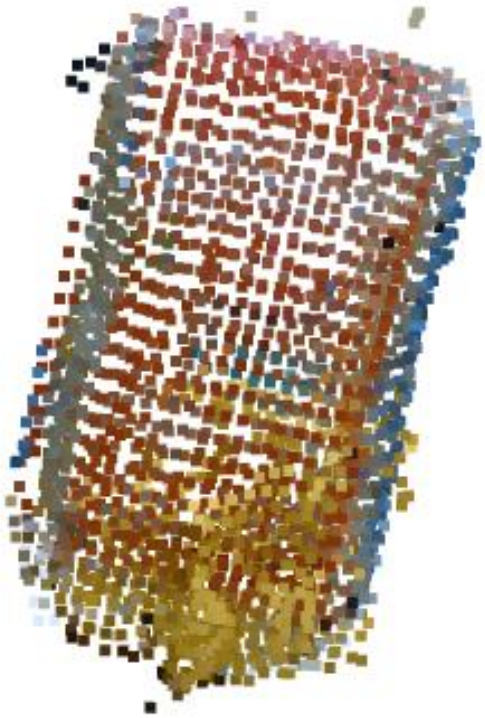
Željeni oblak točaka se smanji i spremljen je u pametni pokazivač. Funkcija funkcionira tako da se točke originalnog oblaka točaka spajaju u voksele, a koliko točaka će se spojiti u jedan voksel ovisi o veličini voksela. Rezultat je da je oblak točaka razjređen, ali i dalje dovoljno reprezentativan i najbitnije znatno manje veličine (memorije). Za primjer je uzet oblak točaka rekonstruirane geometrije Cedevite prikazan na slici ispod.

Slika 46. VoxelDownSample – 1

Ovaj oblak točaka je sastavljen od otprilike 15 oblaka točaka, ima 660000 linija u Notepad-u i zato zauzima 25 MB memorije. Uz primjenu funkcije sa željenom veličinom voksela od 0.005 dobije se oblak prikazan na slici ispod.

Slika 47. VoxelDownSample – 2

Sada je oblak veličine 400 KB što je 60 puta manje memorije i ima manje od 10000 linija u Notepad-u. Koliko točno je optimalno staviti veličinu voksela ne može se generalno reći. Ovisi o oblaku točaka koji vam kamera dohvaća (rezolucija kamere) i ovisi s kojim izgledom oblaka je korisnik zadovoljan i za koju primjenu se koristi smanjeni oblak. Slijedi primjer istog početnog oblaka točaka uz veličinu voksela 0.009. Odmah je jasno da će oblak izgledati rjeđe jer se povećala veličina voksela.



Slika 48. VoxelDownSample – 3

Sada oblak točaka ima oko 2500 linija u Notepad-u i zauzima memorije 105 KB. Sada je svaki voksel jasno vidljiv, te je i dalje vidljiva kontura Cedevite. Ova funkcija gotovo uvijek se koristi kao prvo procesuiranje koje se napravi nad oblakom točaka jer znatno olakšava daljnje procesuiranje. Koristi se kao priprema oblaka točaka za kompliciranije primjene kao što je ICP. Bitno je razumjeti da čak i ovaj primjer oblaka točaka je zapravo dobar i često se ovakvi 'jako' točkasti oblaci koriste dalje u post-processingu. Moguće je napraviti zahtjevnu funkciju kao ICP na ovakvo malim oblacima točaka i zatim dobivenu rotaciju i translaciju primijeniti na originalne oblake točaka, zato što ICP je zahtjevna funkcija dok rotacija i translacija su funkcije koje ne zahtijevaju puno resursa. Puno brže je odraditi voksel na dva oblaka i ICP na ta dva oblaka i taj rezultat iskoristiti na originalnim oblacima, nego što bi bilo odmah ICP na dva originalna oblaka. Prvi navedeni način bi trajao nekoliko sekundi, a drugi vjerojatno nekoliko minuta ako se koristi bez pomoći grafičke kartice.

6.3. Segmentacija pozadine u oblaku točaka

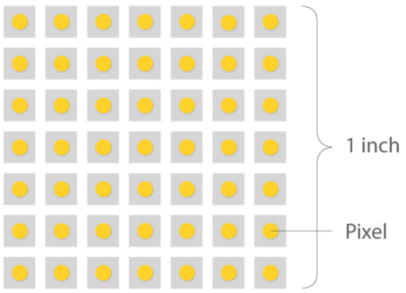
Proces segmentacije pozadine u 2D fotografijama je relativno jednostavan primjenom funkcije iz OpenCV knjižnice naziva *inRange()*. Da bi se segmentirala pozadina iz oblaka točaka potrebno je neko razumijevanje zapisa 2D i 3D podataka. Na slici 43. Prikazan je zapis oblaka točaka, i rečeno je kako je 2D slika jako sličan zapis oblaku točaka samo bez Z osi. Primjer na kojem će biti prikazana segmentacija može se vidjeti na slici 49, gdje je prikazana 2D fotografija s pripadajućim oblakom točaka, znači to je isti uzeti frame samo je lijevo prikaz u fotografiji (2D), a desno prikaz u oblaku točaka (3D).

Slika 49. 2D fotografija s pripadajućim 3D oblakom točaka

Na fotografiji se nalazi predmet, žuta pozadina od žutog papira, žuta rukavica i dio ruke. Cilj je da u oblaku točaka ostane samo željeni predmet. Zašto? Opet je razlog ICP za koji je optimalno da u oblaku točaka bude samo predmet i što bolje kvalitete. Pozadina može imati i značajnu ulogu strojnom učenju. Iako se u ovome radu za strojno učenje koriste samo 2D fotografije, jedan od mogućih nastavaka na ovaj rad bilo bi strojno učenje koje će uzimat u obzir i oblake točaka. Korištenje 2D podataka za duboko učenje se koristi godinama, dok je korištenje 3D podataka relativno novo i koristi se tek nekoliko godina unazad, najviše u automobilskoj industriji.

6.3.1. Segmentacija žutih točaka (SelectedByIndex)

Ako je oblak točaka isto što i 2D fotografija samo uz dodanu Z os, jasno je da će imati isti broj točaka. Jednostavno kamera ima određenu rezoluciju (broj piksela) koji uslika. Kao na slici 50., označeni piksel je isti u 2D i 3D uz razliku dodane Z osi u 3D prikazu.



Slika 50. Piksel 2D [11]

Ranije je pisano o broju redova oblaka točaka u Notepad-u, to je broj točaka, broj piksela odnosno broj indeksa. 2D fotografija i 3D oblak točaka trebali bi imati isti broj indeksa. Na temelju ovoga saznanja radi se ista segmentacija na oba tipa podataka. Nekoliko puta je naglašeno da je obrada 2D podatak znatno lakša, zato se ovdje prvo segmentira pozadina na 2D fotografiji, detaljno je objašnjeno u sljedećoj cjelini, ali uglavnom, kada se dobije 2D fotografija na kojoj je izbačena željena pozadina i pronađe se koji su to točno indeksi, tada se ti isti indeksi izbace iz oblaka točaka funkcijom *SelectByIndex()*, koja ima jedan ulazni argument – vektor koji sadrži potrebne indekse:

```
cloudPtr = point_cloud.SelectByIndex(vector_of_indices);
```

Slika 51. Snippet sintaksa *SelectByIndex()*

Na slici 52. prikazan je segmentirani oblak točaka na kojem je vidljivo kako je većina pozadine uspješno segmentirana. Očekivano dio u kojem se nalazi ruka ostao je vidljiv, a dio ispod proizvoda koji je ostao vjerojatno ima tamniju žutu boju koja nije bila obuhvaćena spektrom žute boje koji je postavljen da se segmentira.

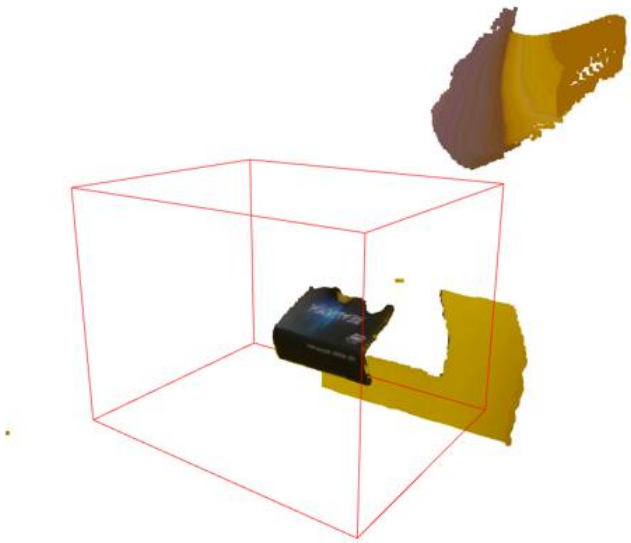
Slika 52. Segmentirani oblak točaka

6.3.2. Izrezivanje oblaka točaka (Crop)

Ako se točno zna gdje će se predmet nalaziti u odnosu na kameru, tada se može primijeniti funkcija izrezivanja oblaka točaka. U ovom slučaju postav na kojem se nalaze kamere je poznat i točno područje gdje dolaze predmeti je također poznat, tako da su sve prepreke za ovu funkciju eliminirane. Da bi se ova funkcija mogla koristiti treba prvo označiti koje je to područje koje će ostati sačuvano, a izvan toga područja se sve briše. To je napravljeno pomoću *AxisAlignedBoundingBox()* funkcije. Potrebno je definirati raspon geometrije u sve tri osi, i potrebno je definirati boju, koja je naravno u r, g, b obliku. U ovom primjeru je za r, g, b postavljeno 1,0,0 što znači da će boja biti crvena. Rezultat funkcije *Crop()* sprema izrezani oblak točaka u pametni pokazivač, i funkcija ima samo jedan ulazni argument, u ovom slučaju pravokutnik sa željenim koordinatama. Ispod je prikazana prvo sintaksa u C++ programskom jeziku te zatim vizualiziran oblak točaka s 'kutijom' za izrezivanje.

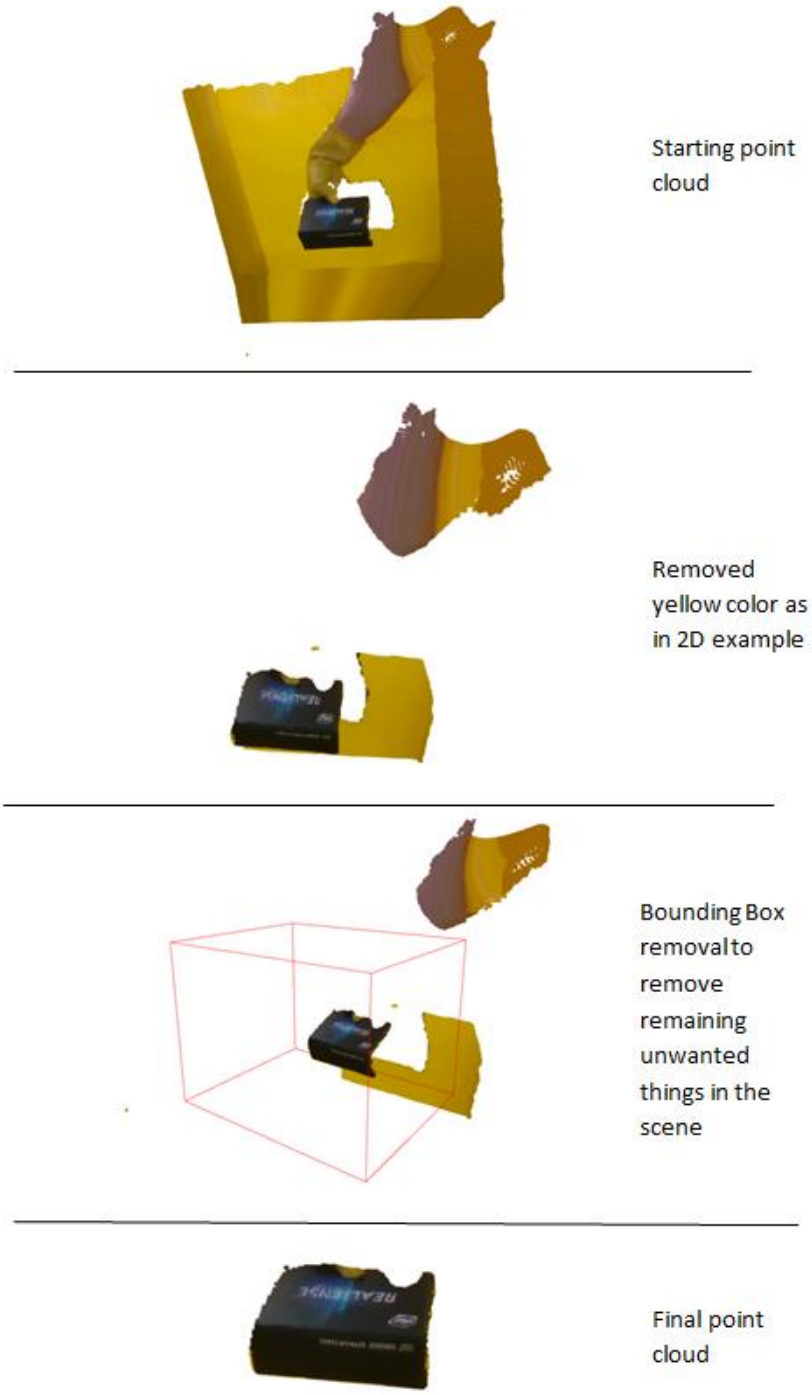
```
open3d::geometry::AxisAlignedBoundingBox box;
box.min_bound_[0] = -0.17;
box.min_bound_[1] = -0.15;
box.min_bound_[2] = 0.2;
box.max_bound_[0] = 0.20;
box.max_bound_[1] = 0.13;
box.max_bound_[2] = 0.5;
box.color_[0] = 1;
box.color_[1] = 0;
box.color_[2] = 0;
cloudPtr = point_cloud.Crop(box);
```

Slika 53. Snippet funkcije *Crop()*



Slika 54. Oblak točaka nakon izrezivanja

Nakon rezanja, u oblaku točaka ostaje samo željeni predmet koji sada zauzima znatno manje memorije – lakši je za dalje procesuiranje, maknuta je pozadina – sav fokus je na željenom predmetu. Slijedi slika koja će ukratko ponovno prikazati cijeli postupak obrade oblaka kako bi on došao u optimalno stanje, od početnog oblaka na kojem je prvo izvršeno *DownSample()*, zatim SelectByIndex() i na kraju *Crop()*.



Slika 55. 'Timeline' jednog oblaka točaka – priprema za ICP

6.4. Primjena stereo kalibracije na oblake točaka

Da se kratko ponovi, stereo kalibracija je prebacivanje iz koordinatnog sustava jedne kamere u koordinatni sustav druge. Stereo kalibracija je u ovom radu bitna za 3D rekonstrukciju, bez nje ne bi bilo moguće napraviti rekonstrukciju. Stereo kalibracija će sada biti prikazana na dva primjera, jedan je primjer Cedevite, a drugi primjer kutije u kojoj dolazi Intel-ova kamera.

Prvi primjer je Cedevita. Na slici ispod prikazane su 2D fotografije uzete sa sve tri kamere, vrlo je jasno koja fotografija je uzeta sa srednje kamere, koja s lijeve i koja s desne.

Slika 56. 2D prikaz Cedevite sa svih kamera

Ako se uzmu oblaci točaka ovih istih fotografija, na njih se primjeni procesuiranje ranije objašnjeno, tako da ostane samo ciljani predmet u oblaku, i sve tri oblaka točaka se prikažu u istom 'viewer-u' dobije se:

Slika 57. Oblaci točaka sa sve tri kamere prikazani zajedno

Iz ovih oblaka točaka vidi se prostorni odnos kamera. A stereo kalibracijom se dobije točno taj odnos u matričnom zapisu. Evo kako postupak izgleda u C++ kodu: prvo je potrebno točno definirati varijable, zatim se iščitava rezultat stereo kalibracije što se radi pomoću *FileStorage()* funkcije iz OpenCV, ali se taj podatak mora spremiti u *cv::Mat* tip varijable koja je također iz OpenCV knjižnice, tek onda se pomoću *for* petlje mogu broj po broj spremati na svoje mjesto u transformacijskoj matrici koja se zatim koristi za stereo kalibraciju. Ovaj kod pokazuje pripremu transformacijske matrice s kamere dva na kameru jedan, a isti postupak se ponovi za kameru tri na kameru jedan, što nema potrebe pokazivati ovdje jer je postupak istu, samo drugo ime varijabli. Slijedi kod.

```
Eigen::Matrix4d transformMatrix21;
Mat matrica1;
cv::FileStorage storage1("/home/ubuntu-z2/RobustVision_New/Cam2-1_transform.yml", cv::FileStorage::READ);
storage1["transformMatrix2-1"] >> matrica1;
for (int i = 0; i < 4; i++)          //Rotaciju unesti u transform_2
{
    for (int j = 0; j < 4; j++)
    {
        transformMatrix21(i,j) = matrica1.at<float>(i,j);
    }
}
cout << "matrica1: " << endl << transformMatrix21 << endl;
storage1.release();
```

Slika 58. Snippet čitanja i upisivanja transformacijske matrice u željenu varijablu

Točne brojčane vrijednosti transformacijskih matrica prikazane su na slici ispod.

$$T_{2-1} = \begin{bmatrix} 0.720039 & 0.019626 & -0.693655 & 0.261786 \\ -0.110655 & 0.990056 & -0.086851 & 0.071492 \\ 0.685053 & 0.139293 & 0.715052 & 0.064748 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Slika 59. Transformacijska matrica kamere 2 na kameru 1

$$T_{3-1} = \begin{bmatrix} 0.785803 & 0.119942 & -0.606736 & 0.239193 \\ -0.009941 & 0.983338 & 0.181515 & -0.083799 \\ 0.618397 & -0.136603 & 0.773902 & 0.098238 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Slika 60. Transformacijska matrica kamere 3 na kameru 1

U Open3D transformacijska funkcija ima sljedeću sintaksu.

```
point_cloud2.Transform(transformMatrix21);
point_cloud3.Transform(transformMatrix31);
```

Slika 61. Snippet transformacija oblaka točaka

Primjeni li se to na lijevi i desni oblak točaka sa slike 56., dobiju se oblaci točaka koji su svi u koordinatnom sustavu prve kamere.

Slika 62. Primjer Cedevita nakon stereo kalibracije

Vidljivo je kako se stranice Cedevite nisu savršeno preklopile. Drugi primjer je kutija za Intel-ove kamere. Ovdje su opet prvo obrađeni oblaci točaka sa svim ranije navedenim funkcijama za smanjenje, segmentaciju i izrezivanje.

Slika 63. Intel kutija prije stereo kalibracije

Slika 64. Intel kutija nakon stereo kalibracije

Na oba primjera se može primijetiti kako stranice nisu savršeno preklopljene. Jer u teoriji u savršenom svijetu i savršenim uvjetima stereo kalibracija bi savršeno odredila potrebnu rotaciju i translaciju da se dobiju preklopljeni bridovi. Mogući razlozi zašto ima određenih odstupanja kod stereo kalibracije su:

1. šahovnica koja se koristi za dobivanje stereo kalibracije nije savršeno ravna
2. promjena temperature i vlage u okolini bi mogao biti jedan od faktora
3. šahovnica je bila dimenzija 7x9, ima nekih članaka koji ukazuju da bi bilo bolje koristiti što više šahovnicu većim dimenzija kako bi bilo što više kuteva za detektirati na slici, npr. dimenzija blizu 20x20
4. neravnomjerna raspodjela svjetlosti i prodiranje vanjske svjetlosti s jedne strane postava je bi mogao biti najbitniji faktor.

6.5. Primjena ICP na oblake točaka

Zato što stereo kalibracija ne daje savršene rezultate ide se dalje u post-processing oblaka točaka. Tijekom ovog rada nekolicinu puta je već spomenuta operacija ICP. ICP je algoritam koji se koristi za minimiziranje razlike između dva oblaka točaka. ICP se često koristi za rekonstrukciju 2D ili 3D površina iz različitih skeniranja, za lokalizaciju robota i postizanje optimalnog planiranja puta itd. Znači ICP ima raznih primjena a jedna od njih je, evo, i rekonstrukcija 3D geometrije proizvoda. I PCL i Open3D knjižnice nude ICP registraciju, ali opet u Open3D je to značajno jednostavnije provesti. Nakon što su se oblaci točaka obradili

na sve ranije navedene načine zaključno s primjenom stereo kalibracije, vrijeme je za ICP. U ovoj funkciji ima nekoliko bitnih varijabli:

1. *threshold* – granica udaljenosti na kojoj će se tražiti točka koja se preklapa
2. *max_iteration* – maksimalni broj iteracija, broj koliko iteracija će ICP algoritam napraviti
3. *relative_fitness* - daje podatak u području preklapanja između cloudova, maksimalna vrijednost je 1
4. *relative_rmse* – rmse znači Root Mean Square Error, pokazuje pogrešku, što niža vrijednost je bolja
5. *transformMatrix* – gruba procjena transformacije, ovo u danoj primjeni ne igra ulogu i najbolje je napraviti običnu dijagonalnu 4x4 matricu kako je vidljivo i u kodu
6. *TransformationEstimationPointToPlane()* – predstavlja tip ICP algoritma. Postoje još *TransformationEstimationForColoredICP, TransformationEstimationForGeneralizedICP i TransformationEstimationPointToPoint.* Sve mogućnosti su isprobane a *PointToPlane()* na ovim primjerima daje najbolje rezultate.

Ako algoritam dostigne vrijednost relativnog *fitnessa* ili relativnog *rmse,* algoritam staje, a ako ne dostigne ni jedno od toga dvoje onda će ići do postavljenog broja iteracija *max_iteration*. Slijedi C++ *snippet* koda za ICP.

```
    /***   ICP   ***/
cloud1.EstimateNormals(open3d::geometry::KDTreeSearchParamHybrid(0.001 * 2.0, 30));
cloud2.EstimateNormals(open3d::geometry::KDTreeSearchParamHybrid(0.001 * 2.0, 30));

double threshold = 0.005;
cout << "Kreće ICP:" << endl;
open3d::pipelines::registration::ICPConvergenceCriteria crit;
crit.max_iteration_=250;
crit.relative_fitness_=1.000000;
crit.relative_rmse_=1.000000e-06;

Eigen::Matrix4d transformMatrix;
transformMatrix.setZero(4,4);
transformMatrix(0,0) = 1;
transformMatrix(1,1) = 1;
transformMatrix(2,2) = 1;
transformMatrix(3,3) = 1;

open3d::pipelines::registration::RegistrationResult reg21 =
    open3d::pipelines::registration::RegistrationICP(cloud2, cloud1,
        threshold, transformMatrix,
            open3d::pipelines::registration::TransformationEstimationPointToPlane(), crit);

cout << "ICP transformacijska matrica: " << endl << reg21.transformation_ << endl << endl;
cout << "Fitness: " << reg21.fitness_ << endl << endl;
cout << "rmse: " << reg21.inlier_rmse_ << endl;

cloud2.Transform(reg21.transformation_);
```

Slika 65. Snippet ICP

Bitno je napomenuti kako *TransformationEstimationPointToPlane()* zahtjeva još da oblak točaka sadrži normale na svoje točke, bez ovoga ova metoda ne radi dok druge metode mogu raditi i bez ovoga. Ovo je jedan od razloga zašto ova metoda daje bolje rezultate od ostalih. Naredba *EstimateNormals()* vidljiva je na početku *snippeta*. Prikazani kod prikazuje ICP algoritam primijenjen tako da se oblak točaka 2 rotira i translatira da se iste točke preklope s oblakom točaka 1. I zadnja linija koda prikazuje onda primjenu rezultata ICP-a, transformacijske matrice, na oblak 2. Isti postupak se ponovi za transformaciju oblaka točaka 3 na oblak točaka 1. Slijedi rezultat ICP na primjeru Cedevite na sljedećoj slici.

Slika 66. Cedevita nakon primjene ICP registracije

Oblaci točaka sada su lijepo preklopljeni. Vidljivo je i nešto šuma, ali to je već ranije objašnjeno kako ove kamere vjerojatno ne mogu uzeti bolje oblake točaka od ovoga. ICP na ovom primjeru na temelju iste srednje plohe Cedevite može lijepo preklopiti sva tri oblaka točaka. Logično, ako oblaci točaka nemaju zajedničkih točaka nema se što preklapati i ICP registracija nema na temelju čega dati rezultate. Ovdje se postavlja pitanje zašto se ICP nije koristio odmah bez stereo kalibracije? Odgovor je da ICP može funkcionirati samo ako su oblaci točaka već dovoljno blizu jedan drugoga. Ako su udaljeni kao npr. prije stereo kalibracije, ICP ne bi dao nikakav rezultat.

6.6. Rekonstrukcija 3D geometrije proizvoda

Postupak rekonstrukcije kompletne geometrije proizvoda ima sljedeću logiku: predmet staviti pred kamere i uzeti oblak točaka, pričekati da se predmet oko svoje verikalne osi okrene za otprilike 20° i opet uslikati oblak točaka, i tako dok se cijeli predmet ne izrotira, što bi za slučaj da se slika svakih 20° bilo 18 oblaka točaka. Rezultat je 18 oblaka točaka sa svake kameri što je ukupno 54 oblaka točaka. Postupak nakon uzetih oblaka točaka je sljedeći:

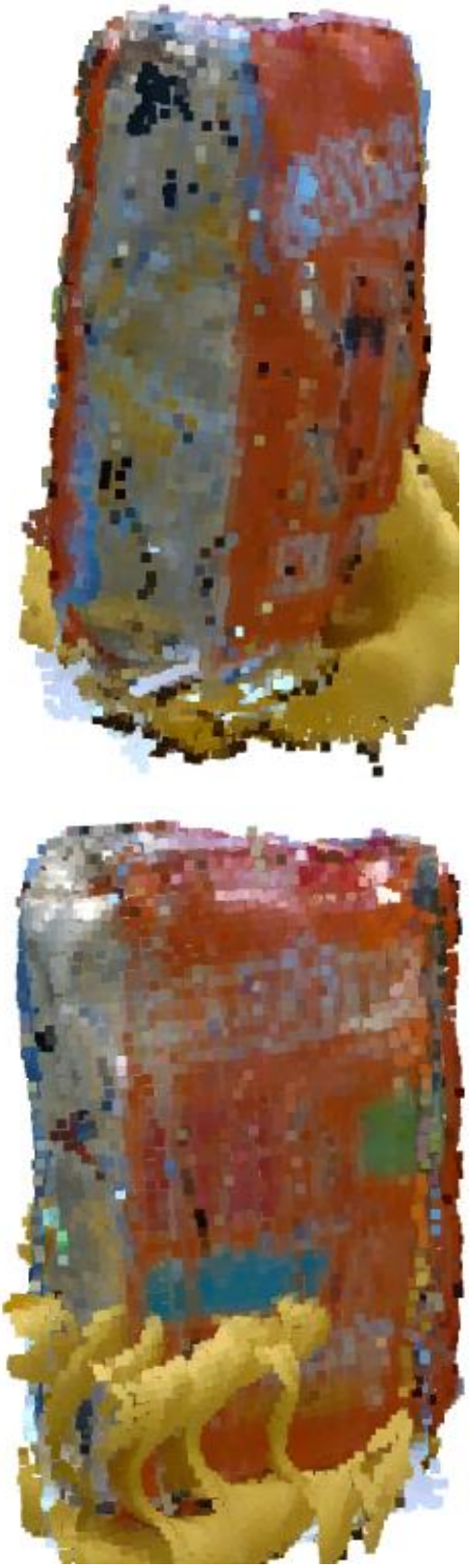
1. Primijeniti *VoxelDownSample()* na sve oblake točaka
2. Primijeniti *SelectByIndex()* na sve oblake točaka
3. Primijeniti *Crop()* na sve oblake točaka

4. Na prvi oblak točaka sa svake kamere primijeniti stereo kalibraciju i zatim ICP i to spremiti u jedan oblak točaka. Dobije se jedan oblak kao na slici 66, tako napraviti za svih 18
5. Primijeniti *VoxelDownSample()* na svih novih 18 oblaka točaka, jer im se povećala veličina zbog zbrajanja 3 oblaka točaka u jedan
6. Sada ide ICP tih 18 novih oblaka. Kreće se s prvim oblakom točaka u odnosu na drugi, pa drugi na treći, treći na četvrti itd. Završiti će se na kraju u koordinatnom sustavu zadnjeg oblaka točaka. Opcija je završiti u koordinatnom sustavu prvog ili zadnjeg, svejedno je što se odabere, u ovom radu se radilo da završi u zadnjem. Ovo znači da će se prvi oblak morati transformirati kroz 17 koordinatnih sustava da bi došao do zadnjega. Drugi oblak će proći kroz 16, treći kroz 15 itd.

```
/***  ICP ***/
    double threshold = 0.005;
    cout << "Kreće ICP:" << endl;
    open3d::pipelines::registration::ICPConvergenceCriteria crit;
    crit.max_iteration_=100;
    crit.relative_fitness_=1.00000;
    crit.relative_rmse_=1.000000e-06;

    for(int i=1; i<14;i++)
    {
        point_cloud1.Clear();
        point_cloud2.Clear();
        open3d::io::ReadPointCloud("cede_tris_" +to_string(i) + ".ply", point_cloud1);
        open3d::io::ReadPointCloud("cede_tris_" +to_string(i+1) + ".ply", point_cloud2);
        cloudPtr1 = std::make_shared<open3d::geometry::PointCloud>(point_cloud1);
        cloudPtr2 = std::make_shared<open3d::geometry::PointCloud>(point_cloud2);
        point_cloud1.EstimateNormals(open3d::geometry::KDTreeSearchParamHybrid(0.01 * 2.0, 30));
        point_cloud2.EstimateNormals(open3d::geometry::KDTreeSearchParamHybrid(0.01 * 2.0, 30));

        open3d::pipelines::registration::RegistrationResult reg1 =
                    open3d::pipelines::registration::RegistrationICP(
                        point_cloud1, point_cloud2, threshold, transformMat,
                        open3d::pipelines::registration::TransformationEstimationPointToPlane(), crit);

        cout << "fitness" + to_string(i) + ": " << reg1.fitness_ << endl;
        cout << "rms" + to_string(i) +": " << reg1.inlier_rmse_ << endl;

        transformations[i] = reg1.transformation_;
    }

/*** Transformacije cloudova ***/
    for(int i=1; i<14;i++){
        point_cloud1.Clear();
//          naziv_tris1 = "Cedevita_3D" + "/Trises" + "/tris_br_" + to_string(i) + ".ply";   //   t = 4.455e-06 s
        open3d::io::ReadPointCloud("cede_tris_" + to_string(i) +".ply", point_cloud1);

        for(int j=0+i; j<14; j++){
            point_cloud1.Transform(transformations[j]);
        }

        pc_sum = pc_sum + point_cloud1;
    }
/*** Prikaz clouda ***/
    std::shared_ptr<open3d::geometry::PointCloud> cloudPtr21 = std::make_shared<open3d::geometry::PointCloud>(pc_sum);
    cloudPtr21 = pc_sum.VoxelDownSample(0.002);   //  t = 0.00780749 s
    open3d::visualization::DrawGeometries({cloudPtr21}, "Zbrojeni u jedan cloud the end"
                                          1500, 1000, 150, 50, false, false, false, &look, &up, &front, &zoom);
```

Slika 67. Snippet postupka za 3D rekonstrukciju geometrije proizvoda

Na *snippetu* iznad je prikazano u *for* petlji kako se radi ICP svih oblaka točaka i posprema se u varijablu *transformations*, da se zatim u sljedećoj *for* petlji učitavaju oblaci točaka i transformiraju koliko puta je potrebno. Ovo je primjer u kojem se uzelo 14 oblaka točaka što znači da je kut kod uzimanja oblaka točaka bio nešto više od 20°. U varijablu *pc_sum* zbroje se svi oblaci točaka, još jednom se napravi *VoxelDownSample()* i rekonstruirana je 3D geometrija proizvoda koja je prikazana na slici ispod. Vidljivi su neki šumovi i nedovoljna segmentacija rukavice na donjem dijelu proizvoda. Potencijalno rješenje bilo bi robotom hvatati predmet npr. vakuumskom hvataljkom koja bi bila obučena u žutu boju i npr. uhvatila predmet s gornje strane i samo ga rotirala oko vertikalne osi kako je ranije napisano. I druga bitna stvar je imati veću kontrolu nad svijetlom. U ovom labosu svjetlost se mijenja tijekom dana zbog velikog utjecaja dnevnog svjetla na postav, čak i kada padne mrak nejednolik je raspored svijetla po postavu te na svaku kameru svjetlost pada pod drugim kutem što znatno otežava segmentiranje boje iz slike, posljedično i rekonstrukciju predmeta, a pokazat će se u sljedećem poglavlju i generiranje 2D seta slika.



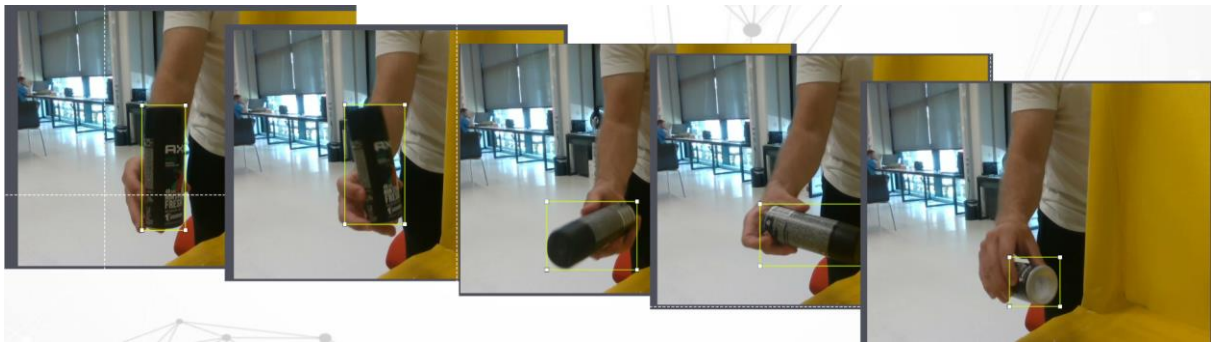
Slika 68. Rekonstruirana 3D geometrija proizvoda Cedevite

7. GENERIRANJE SETA 2D FOTOGRAFIJA ZA TRENIRANJE NEURONSKIH MREŽA

3D rekonstrukcija geometrije proizvoda bila je jedan od dva glavna zadatka ovoga rada. Drugi glavni zadatak je automatski generirati set fotografija za trening neuronskih mreža. 2D podaci se obrađuju u OpenCV open-source knjižnici koja je relativno laka za korištenje. Set slika za treniranje neuronskih mreža mora biti takav da se na slici označi predmet koji se želi naučiti. Kako automatski označiti proizvod na slici ako računalo ne zna koji proizvod označiti, jer kad bi računalo znalo označiti proizvod to bi značilo da ono već zna prepoznat taj proizvod, ne bi bilo potrebe za učenjem. Tako da, danas je još uvijek najčešći oblik označavanja predmeta na slici manualnim načinom.

7.1. Manualno generiranje

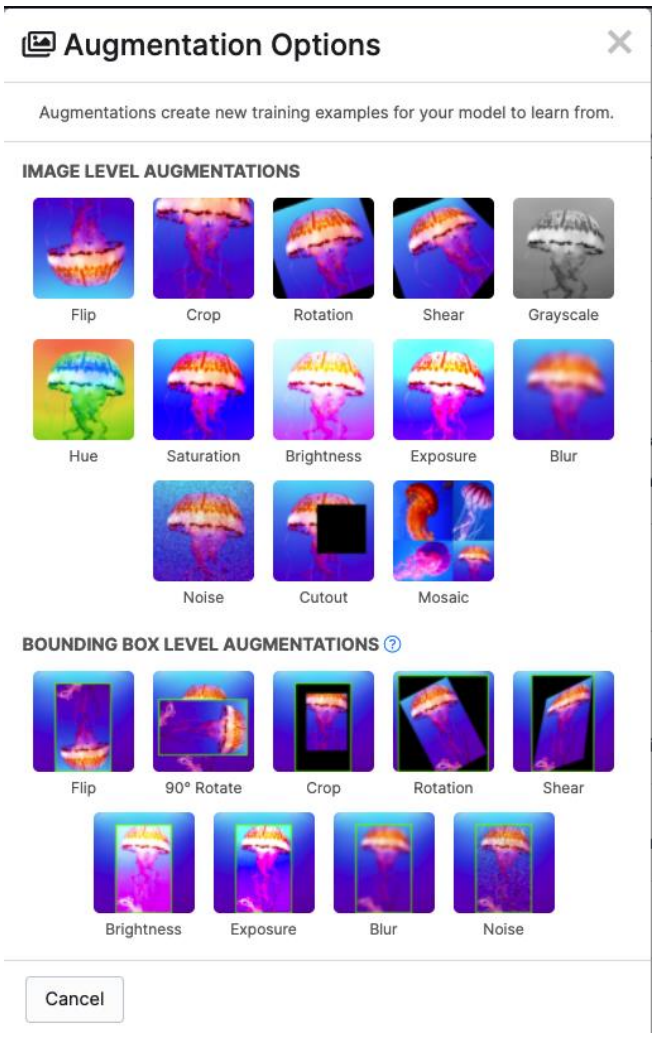
Manualno označavanje slika svodi se na to da se uzme velika količina slika i u nekom od programa za označavanje se svaka slika manualno označi. Isprobano je manualno označavanje u besplatnom programu dostupnom na internetu imenom *roboflow*. Program je dobar, lak za korištenje i nakon označenih svih slika lako se generiraju podaci spremni za korištenje u željenoj verziji YOLO neronske mreže. Na slici ispod su prikazane neke od slika na kojima je predmet koji želimo da mreža nauči manualno označen.



Slika 69. Manualno označavanje predmeta

U ovom setu slika nalazilo se oko 400 slika i za ručno označivanje trebalo je oko 3 sata posla, a u nekim profesionalnim primjenama, čest je slučaj da se set sastoji od nekoliko tisuća ili čak stotina tisuća slika, pogotovo ako se mrežu uči prepoznavanje više predmeta. I sve se te slike moraju ručno označiti. Roboflow je dobar zato što nudi laku augmentaciju slika, tako da se broj slika u setu može brzo povećati. Augmentacija se radi tako da se na već postojeće slike primjene neke operacije, sve u svrhu kako bi dobili raznolikost i povećali broj slika. Primjer

mogućih augmentacija ponuđenih na *roboflow* stranici prikazan je na slici ispod. *Roboflow* je besplatan do seta od 1000 slika.



Slika 70. Mogućnosti augmentacije u roboflow

Uz manualno označenih 400 slika, dodano je još oko 200 augmentiranih slika koje se lako generiraju, dovoljno je par klikova. Za generiranje 400 slika s 3 kamere koje se koriste u izradi ovoga rada bilo je potrebno oko 5 sekundi, uz još 3 sata potrebnih za označavanje, to je vrijeme da bi se jedan predmet pripremio za učenje neuronske mreže.

Slika 71. Manualno označeni set slika spreman za treniranje neuronske mreže

7.2. Automatsko generiranje

Ideja automatskog generiranja u ovome radu bila je da je u kratkom vremenu moguće generirati veliki broj slika na kojima se nalazi samo predmet i sve ostalo u kadru je žute boje.

7.2.1. Generiranje velikog broja slika

Ove kamere u mogućnosti su hvatati 30 frameova u sekundi, ali samo spremanje fotografija traje duže te ako bi htjeli hvatati 30 fps i odmah spremati slike, ne bi bilo moguće 30 slika po sekundi, program je jednostavno prespor za to. Rješenje se pronašlo u sljedećem:

1. Hvatanje frameova se odvija zasebno, prvo se frameovi svih kamera spremaju u vektor predviđen za tu kameru
2. Kasnije kada su uzeti svi frameovi, ide daljnje procesuiranje i spremanje slika.

Sljedeći *snippet* prikazuje uzimanje i spremanje frameova 10 sekundi, u tome roku svaka kamera može uzeti 300 frameova što je ukupno 900 slika. To je iznimno brzo generiranje velikog broja slika nekog proizvoda. Naredba *push_back* samo sprema frameove jednog iza drugog u vektor *framesS1*, *framesS2*, *framesS3*. Kasnije se napravi petlja koja će ići do 300 i na svaku sliku se primjeni obrada koja je objašnjena u sljedećim poglavljima.

```
int iter = 1;
std::vector<rs2::frameset> framesS1, framesS2, framesS3;
framesS1.reserve(500);
framesS2.reserve(500);
framesS3.reserve(500);
/****************************** UZIMANJE FRAMEOVA ****************************/
previous_time = std::chrono::steady_clock::now();
while (elapsed_seconds.count() < 10)   // koliko sekundi će se uzimati frameovi
{

    frames1 = pipeReal1.wait_for_frames();
    frames2 = pipeReal2.wait_for_frames();
    frames3 = pipeReal3.wait_for_frames();
    frames1.keep();
    frames2.keep();
    frames3.keep();
    framesS1.push_back(frames1);
    framesS2.push_back(frames2);
    framesS3.push_back(frames3);

    cout << "Broj iteracija: " << iter << endl;
    iter++;
    current_time = std::chrono::steady_clock::now();
    elapsed_seconds = current_time-previous_time;
    cout << elapsed_seconds.count() << endl << endl;
}
```

Slika 72. Snippet generiranje velikog broja slika

Ako je moguće uspješno segmentirati žutu boju, na slici bi ostao samo predmet. U ovakvom slučaju koristi se funkcija iz OpenCV knjižnice naziva *inRange()*.

7.2.2. Funkcija inRange()

Funkcija *inRange()* ima 3 ulazna i 1 izlazni parametar. Ulazni parametri redom su:

1. Ulazna slika koja mora biti pretvorena u HSV model boja
2. Donja granica HSV vrijednosti
3. Gornja granica HSV vrijednosti.

Izlaz funkcije je slika koja će imati vrijednost 255 na elementima gdje element ulazne slike imati vrijednost između postavljene navedene donje i gornje granice HSV vrijednosti, dok će svi ostali elementi imati vrijednost 0. Kod je prikazan na sljedećem snippetu, a ispod snippeta će redom biti objašnjen kod i prikazani rezultati.

```
cvtColor(Image1, hsv_image, COLOR_BGR2HSV);
inRange(hsv_image,Scalar(12,100,75), Scalar(36,255,255),frame_threshold);
bitwise_not(frame_threshold, frame_threshold_not);
bitwise_and(Image1, frame_threshold_not, Image1);
```

Slika 73. Snippet *inRange()*

Početna slika je u varijabli *Image1*, to je klasična slika u BGR formatu boja. Prikazana na slici ispod.

Slika 74. Početna BGR slika

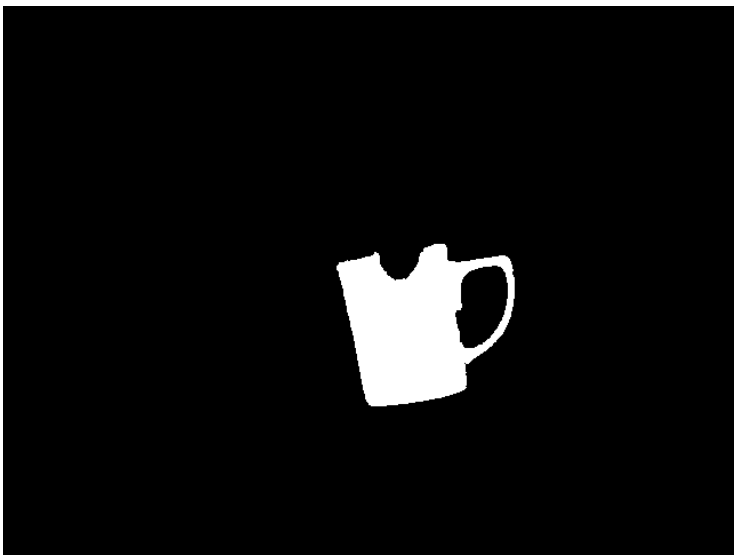
Zatim funkcijom *cvtColor()* slika se prebaci iz BGR modela boja u HSV model boja, jer je to model boja po kojem funkcija *inRange()* radi.

Slika 75. HSV slika

Slika u HSV modelu boja se stavlja kao ulazni parametar u funkciju, i još se određuje donja i gornja granica HSV vrijednosti koje se želi segmentirati. Vrijednosti vidljive u *snippetu* iznad su vrijednosti žute boje u HSV modelu boja. H je prva vrijednost koja predstavlja tri primarne boje – crvenu, plavu i žutu, S je druga vrijednost koja predstavlja intenzitet boje, V je treća vrijednost koja predstavlja koliko je svjetla odnosno tamna boja. *InRange()* na izlazu daje sliku koja sve elemente izvan navedenog raspona ima 0. Na slici su bijele boje elementi koji su unutar zadanih vrijednosti, elementi žute boje odnosno pozadina.

Slika 76. Izlaz *inRange()* funkcije

Kako je za primjenu u ovome radu potrebno maknuti pozadinu, potrebno je zapravo invertirati sliku koja se dobila. To se lako napravi funkcijom *bitwise_not()* koja invertira bitove slike, tamo gdje je 0 bit će 1, a tamo gdje je iznad 1 bit će 0. Rezultat je da sljedeći.



Slika 77. Slika nakon *bitwise_not()*

I za kraj još malo poigravanja s bitovima, funkcijom *bitwise_and()* moguće je staviti dvije slike kao ulaze i za izlaz dobiti sliku koja će sadržavati samo one elemente koji postoje na obje slike. Klasični *AND* operator. Ulazne slike i rezultat prikazani su ispod, gdje se može vidjeti kako je rezultat zapravo prva slika s koje su ostali samo elementi koji su bijele boje na drugoj - koji imaju vrijednost veću od 0.

Slika 78. Rezultat - boja segmentirana

7.2.2.1. 3D segmentacija – dobivanje potrebnih indeksa

U prošloj cjelini je objašnjeno kojom funkcijom se segmentira oblak točaka, ali nije bilo objašnjeno kako se dolazi do vektora koji sadrži indekse na kojima je žuta boja. Slika koja se dobije *bitwise_not* funkcijom je zapravo slika na kojoj su sve pozicije, na kojima je segmentirana žute boja, 0. Prvo je tu sliku potrebno staviti u jedan veliki vektor OpenCV tipa *Mat*. U tom obliku je moguće ovu sliku zapisati u vektor tipa *std::vector* koji je potreban za

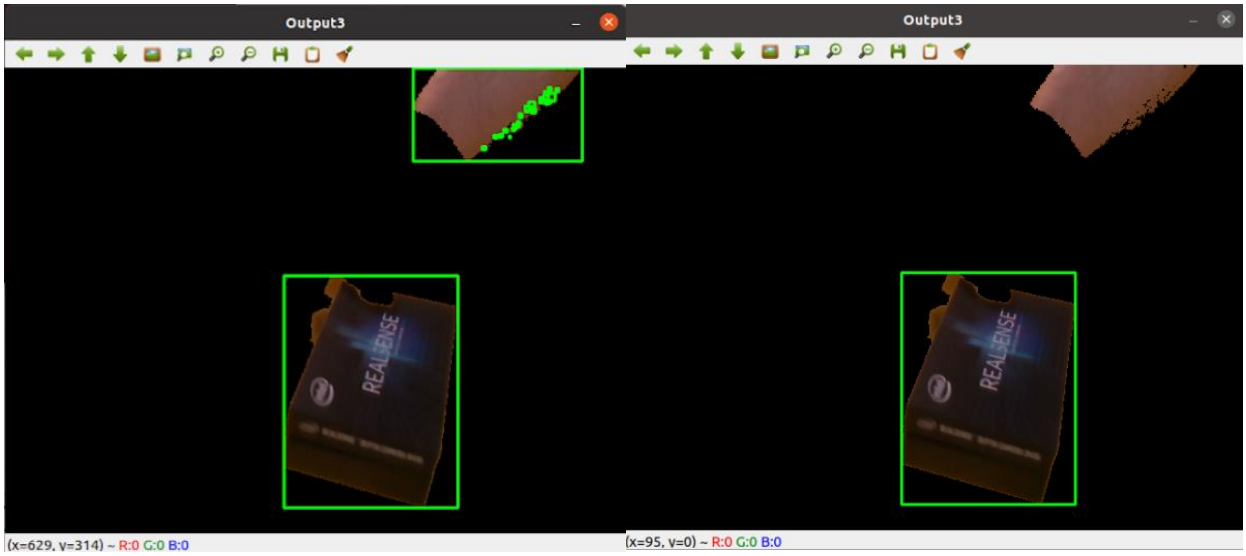
dalje operacije. *For* petljom koja prolazi kroz cijelu sliku koja je sada zapravo u obliku vektora, traže se indeksi koji su ostali na slici (vrijednost 255). Zatim zapišemo mjesto na kojem se taj element nalazi – indeks toga elementa i svaki put kada je pronađen spremi se u rezultantni vektor koji će se iskoristiti za segmentaciju boje u oblaku točaka. Primjer objašnjenog koda nalazi se na sljedećoj slici.

```
cv::Mat img_flat = frame_threshold_not.reshape(1,1); // flat 1d

std::vector <long unsigned int> rez_vect;
std::vector <int> veki;

veki = imgbw_flat.row(0);//    fInitializeDevices();
for (int j = 0; j< veki.size(); j++)
{
    if(veki.at(j) == 255){
        veki.at(j) = 1;
        veki.at(j) = veki.at(j) * j;
        rez_vect.push_back(veki.at(j));
    }
}
```

Slika 79. Snippet koda za vektor izbačenih indeksa

7.2.3. Pronalazak i označavanje kontura

Nastavak je da se pomoću funkcije iz OpenCV knjižnice naziva *findContours()* pronađu sve konture sa slike. Uzmimo za primjer sljedeću sliku da bi se objasnilo na koji način ova funkcija radi.



Slika 80. Primjer pronalaska i iscrtavanja kontura

Funkcija će svaki prijelaz iz 0 u 1 detektirati kao konturu, zato je u desnom gornjem kutu lijeve slike pronađeno više kontura. Ovaj slučaj se može riješiti tako da se isprogramira iscrtavanje samo najveće kontura na slici pa se dobije rezultat prikazan na desnoj slici. Slijedi koda za pronalazak najveće konture na slici.

```
cvtColor(Image1, gray1, COLOR_BGR2GRAY);

std::vector<std::vector<cv::Point>> contours1;
findContours(gray1, contours1, RETR_LIST, CHAIN_APPROX_NONE);
// Finds the contour with the largest area for cam 1
int area = 0;
int idx;
for(int i=0; i<contours1.size();i++) {
    if(area < contours1[i].size())
    {
        area = contours1[i].size();
        idx = i;
    }
}
```

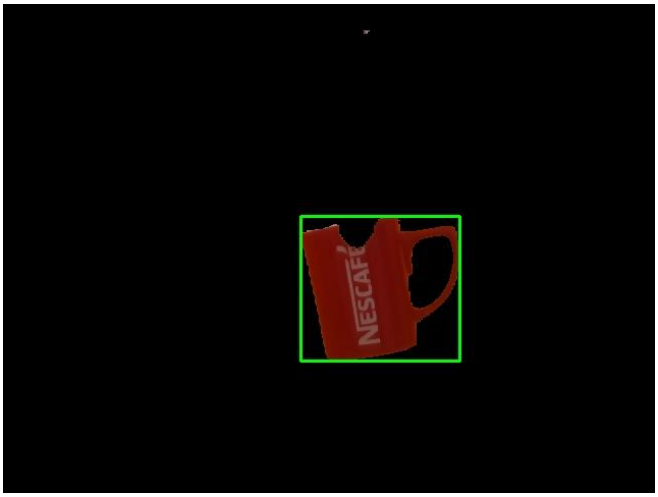
Slika 81. Snippet *findContours()* funkcije

Segmentirana slika se prvo konvertira u *grayscale*, zatim funkcija *findContours()* pronađe sve konture na slici te zatim se preko for petlje pronađe najveća kontura na slici – to se radi kako bi se izbjeglo označavanje nekih točkica koje se nisu dobro segmentirale, ima jedna takva točkica na sredini gore na slici Nescafe šalice koja slijedi. Nakon pronalaska konture, funkcijom *rectangle()* se oko najveće konture iscrtava pravokutnik kojim se zapravo označuje željeni predmet na slici.

```
rect1 = boundingRect(contours1[idx]);
pt1.x = rect1.x;
pt1.y = rect1.y;
pt2.x = rect1.x + rect1.width;
pt2.y = rect1.y + rect1.height;

rectangle(Image1, pt1, pt2, CV_RGB(0 ,255, 0), 2);
```

Slika 82. Snippet *rectangle()* funkcije

Prvo se pronađu koordinate točaka pronađene konture na slici i zatim se u funkciji *rectangle()* postavi slika na koju će se iscrtati pravokutnik čija početna točka ima koordinate spremljene u *pt1* i *pt2*, i zadnja dva argumenta funkcije su boja pravokutnika i debljina linije. Na slici ispod je prikazana označena Nescafe šalica.



Slika 83. Označen željeni predmet

Naravno, istu tu oznaku može se iscrtati i na originalnu početku sliku jer se u istim koordinatama nalazi ta ista šalica.

Slika 84. Označeni željeni predmet na početnoj slici

Pozadina predmeta je iznimno bitna kod dubokog učenja i detekcije predmeta.

7.3. YOLO struktura direktorija

YOLOv5 zahtjeva određenu strukturu ulaznih podataka. Struktura direktorija je prikazana na slici ispod. Struktura direktorija u principu ne bi morala biti točno ovakva, ali ako se mijenja ova struktura moralo bi se mijenjati i neke stvari u kodu neuronske mreže, što su ne potrebne komplikacije.

```
datasets ── penguins ── images ── train
                |          |──────── validation
                |          |──────── test
                |
                |──────── labels ── train
                           |──────── validation
                           |──────── test
```

Slika 85. Struktura YOLOv5 direktorija [13]

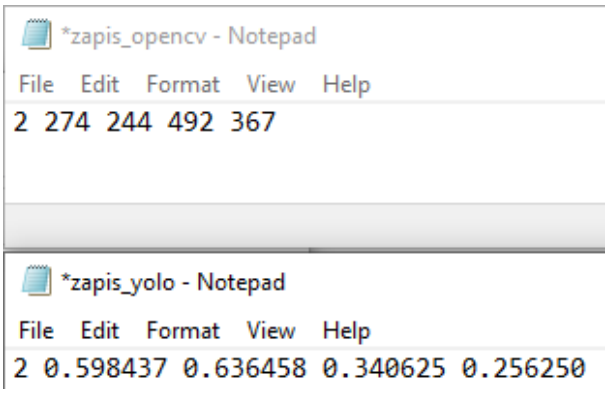
To je što se tiče strukture direktorija, a struktura podataka u tim direktorijima mora biti:

1. Slike u .jpg formatu
2. .txt datoteka **istog** imena kao pripadajuća slika (samo drugog nastavka .jpg/.txt) koja sadrži 2 informacije: klasu predmeta i koordinate pravokutnika za označavanje.

Uzmimo za primjer da je slika u folderu *images/train* imena „P0_cam1nf_1.jpg“, a tekstualna datoteka za tu sliku u folderu *labels/train* mora biti imena „P0_cam1nf_1.txt“.

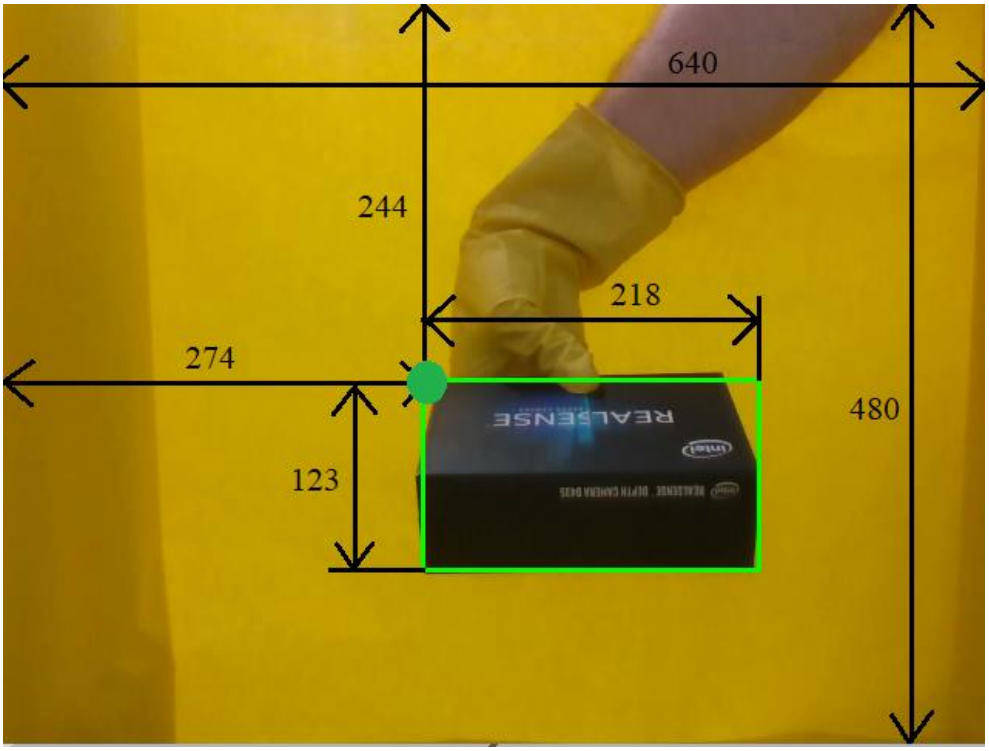
7.3.1. Problem zapisa koordinata oznake predmeta

Do problema dolazi što OpenCV i YOLO na različite načine zapisuju koordinate pravokutnika za oznaku proizvoda na slici. Zapisi je prikazani na slici ispod, gdje prva brojka označava klasu, odnosno broj proizvoda, a ostale 4 brojke označavaju koordinate pravokutnika.



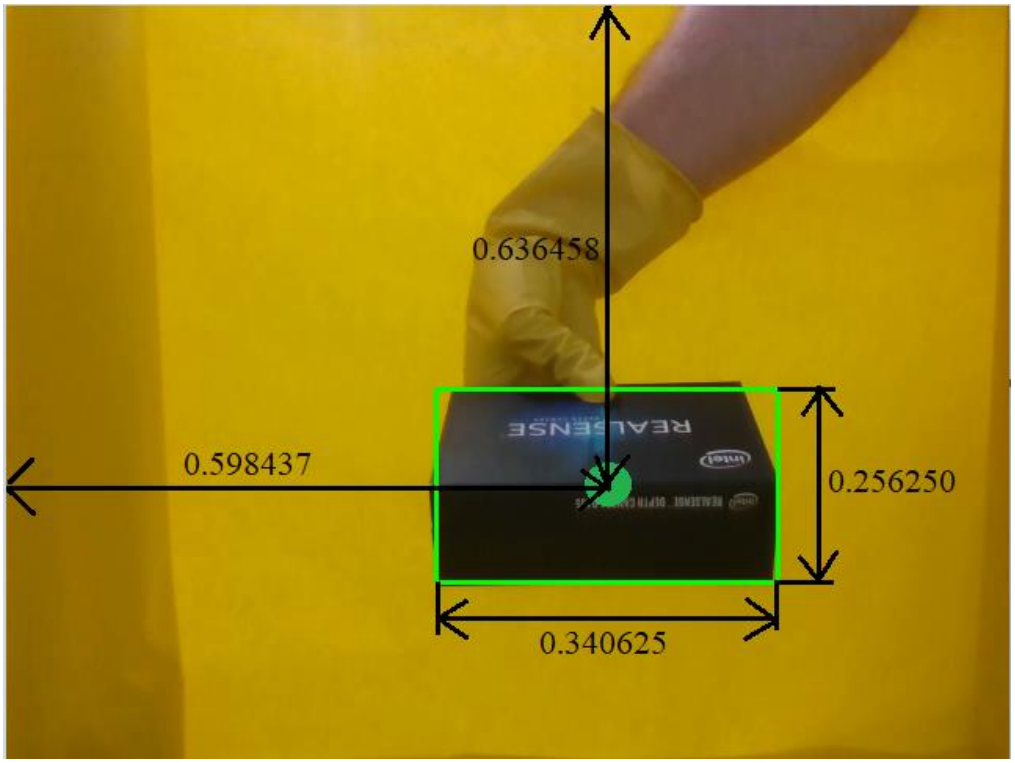
Slika 86. Primjer zapisa koordinata pravokutnika OpenCV vs Yolo

OpenCV nam preko *boundingRect()* daje x i y koordinatu lijeve gornje točke (početne točke) pravokutnika i daje nam duljinu u x i y smjeru. Za pokazni primjer će se uzeti slika ispod, na kojoj su dobivene vrijednosti pokazane na slici iznad (Notepad zapis). Na slici ispod dobivene mjere bile su: početna točka ima koordinate (x, y) = (274, 244), i još duljina u x smjeru iznosi 218 piksela, a duljina u y smjeru 123 piksela. Sve mjere su iskotirane i vidljive na slici.



Slika 87. OpenCV kote pravokutnika

S druge strane, YOLOv5 očekuje zapis u relativnim normaliziranim koordinatama. Mjere su ovdje izražene u postocima u odnosu na veličinu slike. Slika je veličine 640x480.



Slika 88. YOLO kote pravokutnika

Prvi broj u zapisu (nakon broja klase) će biti 0.598437 koji označava da je središte pravokutnika po x osi na 59.8437% slike, gledano s lijeva na desno jer je to pozitivan smjer osi x. Drugi broj u zapisu je 0.636458 koji označava da je središte pravokutnika po y osi na 63.5458% slike, gledano odozgo prema dolje jer je to pozitivan smjer osi y. Treća brojka u zapisu bit će 0.340625 koja označava da je ukupna širina pravokutnika u smjeru osi x 34.0625% ukupne širine slike. Četvrti broj u zapisu je 0.256250 koji označava da je ukupna visina pravokutnika u smjeru osi y 25.6250% ukupne visine slike. Sama matematika pretvorbe iz jednog oblika u drugi nije suviše zahtjevna. Prikazana je na *Snippetu* koji slijedi.

```
classNr = 2;
vr1.bb0 = (rect1.x + (static_cast<float> (rect1.width)/2)) / 640;
vr1.bb1 = (rect1.y + (static_cast<float> (rect1.height)/2)) / 480;
vr1.bb2 = static_cast<float> (rect1.width / 640.0);
vr1.bb3 = static_cast<float> (rect1.height / 480.0);
```

Slika 89. Snippet pretvorba koordinata pravokutnika za YOLO

Sada je još jedino preostalo spremiti tekstualnu datoteku, koja se radi na sličan način kako je ranije bilo pokazano za pohranu intrinzičnih i stereo kalibracijskih parametara.

```
    // /***** Labels save *****/
string filename = "textfile.txt";
ofstream saveloc;
saveloc.open("Kutija_cam1_pic1.txt");
saveloc << to_string(classNr) + " " + to_string(vr1.bb0) + " " +
    to_string(vr1.bb1) + " " + to_string(vr1.bb2) + " " + to_string(vr1.bb3);
saveloc.close();
```

Slika 90. Snippet pohrana tekstualne datoteke

8. TESTIRANJE I EVALUACIJA PRIMJENJIVOSTI PODATAKA ZA TRENIRANJE NEURONSKIH MREŽA

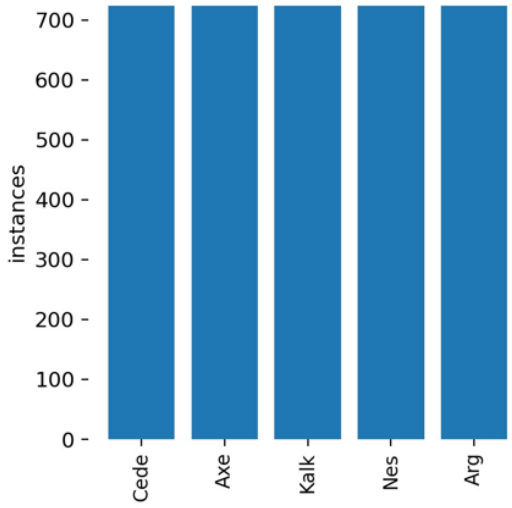
Ovo je najbitnija cjelina gdje će se pokazati je li sve što se prije radilo bilo dobro. Generalno za rad u dubokom učenju sa slikama koriste se konvolucijske neuronske mreže, one su najbolje za ovakav tip podataka. Tijekom godina korištenje neuronskih mreža za učenje i detekciju predmeta išlo je otprilike:

1. R CNN
2. Fast R CNN
3. Faster R CNN
4. YOLO – You Only Look Once.

YOLO je nastao 2015.g. Radi na drugačijem principu od ostalih navedenih neuronskih mreža, koji se pokazao brži i bolji. YOLO omogućuje detekciju predmeta u videima do čak 45 *fps* što je neusporedivo brže od ostalih nabrojanih. Ono gdje YOLO algoritam ima problema je detekcija malih predmeta na fotografiji.

8.1. Funkcioniranje konvolucijskih neuronskih mreža

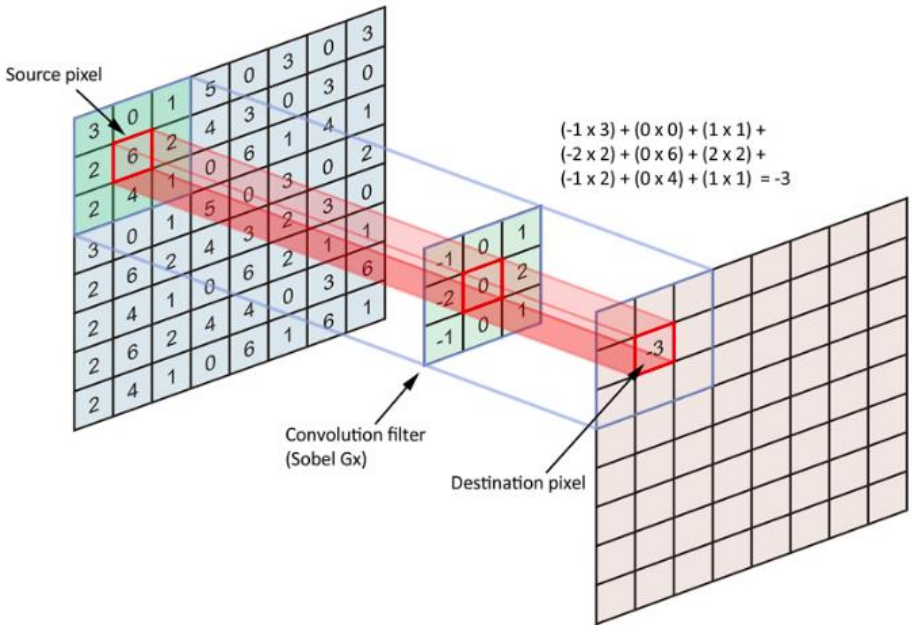
Za početak je bitno dobiti set slika koji je balansirano raspodijeljen – ako se mreža uči detekciji više predmeta, poželjno je da svaki predmet bude na podjednakom broju slika. Ako bi jedan predmet bio znatno češći od drugog, neuronska mreža bi mogla biti pretrenirana na taj predmet i detektirati ga i kad zapravo nije na slici itd. Slika ispod prikazuje raspodjelu seta slika korištenog u sklopu ovog diplomskog rada.



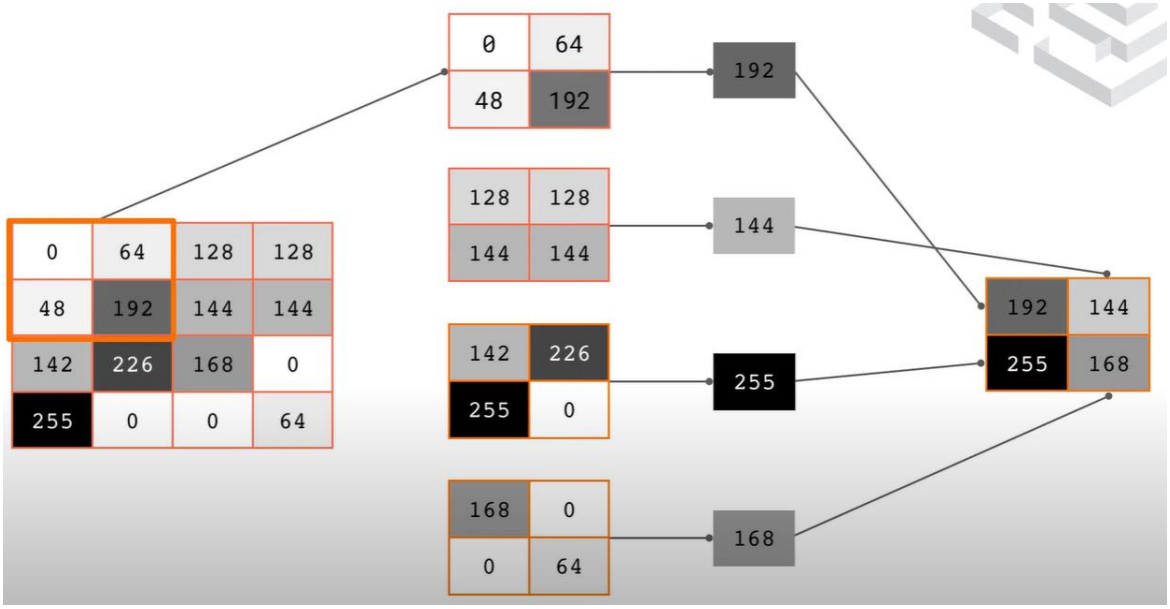
Slika 91. Balansirani set za treniranje neuronske mreže

Dalje unutar same neuronske mreže provode se matematičke operacije kako bi slika postala jasnija samom računalu. Najčešće operacije koje se često koriste i nekoliko puta unutar neuronske mreže na jednoj slici su:

1. Konvolucija – uvijek ima neku jezgru o kojoj ovisi rezultat konvolucije, jezgra prolazi kroz cijelu sliku radeći matematičku funkciju konvolucije
2. Max pooling – česta je primjena 2x2 matrice, ide po slici, te od svakog 2x2 elementa ostaje samo najveća vrijednost

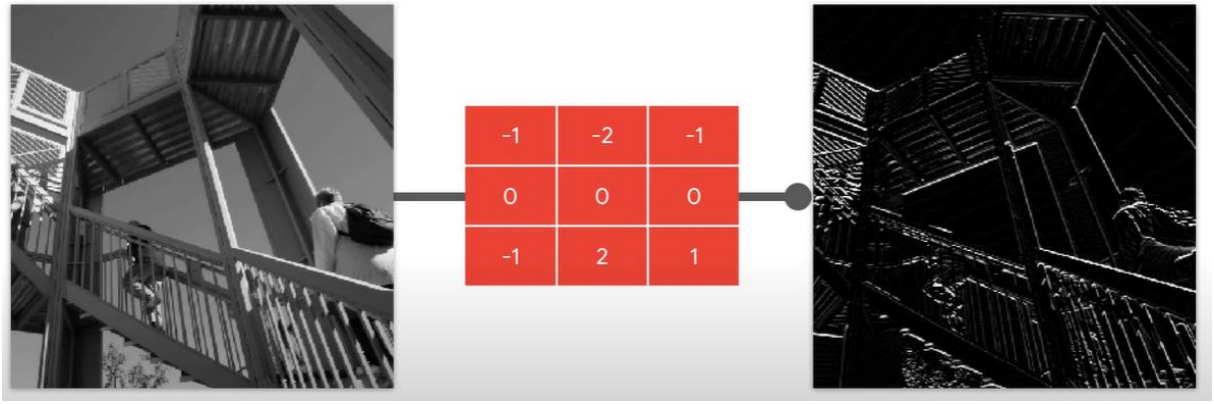


Slika 92. Matematički prikaz konvolucije [13]

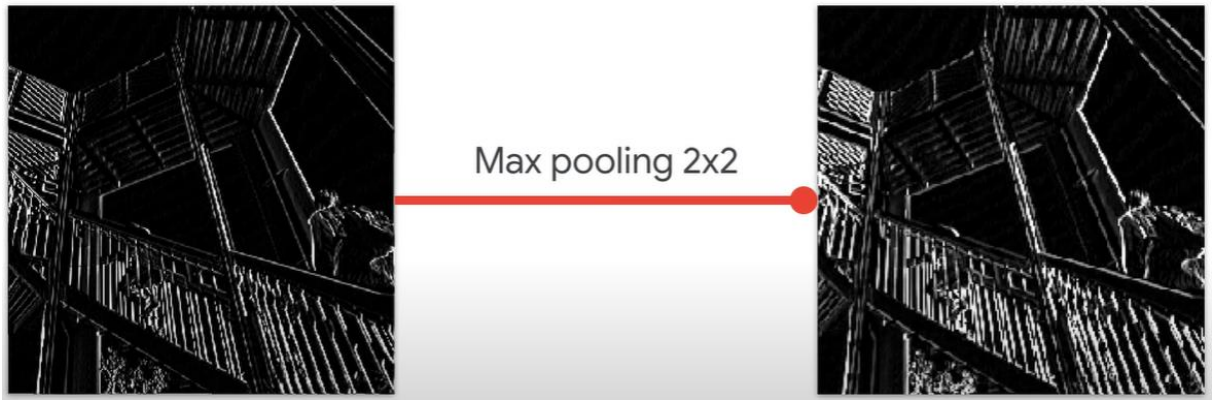


Slika 93. Matematički prikaz Max pooling 2x2 [14]

Slijede konkretni primjeri što napravi konvolucija, a što Max pooling. Svrha je da slika bude razumljivija računalu. Vidljivo je kako nakon konvolucije na slici ostaju bitne konture koje su nakon max poolinga još izraženije.



Slika 94. Primjer konvolucije [14]



Slika 95. Primjer Max pooling [14]

Ove funkcije su određene unutar arhitekture YOLO algoritma, te krajnji korisnik nema veze s njima i nema potrebe puno razmišljati o njima.

8.2. YOLOv5

YOLO je familija arhitektura za detekciju objekata, open-source je i ima dobru popratnu dokumentaciju i potporu. Na YOLOv5 github stranici sve je javno, te se svaki detalj oko ove arhitekture može vidjeti. Ovaj algoritam je napravljen s ciljem brzog i lakog korištenja. Dostupan je za korištenje na razne načine, a neki od najpopularnijih su:

1. Docker
2. Kaggle
3. Google Colab.

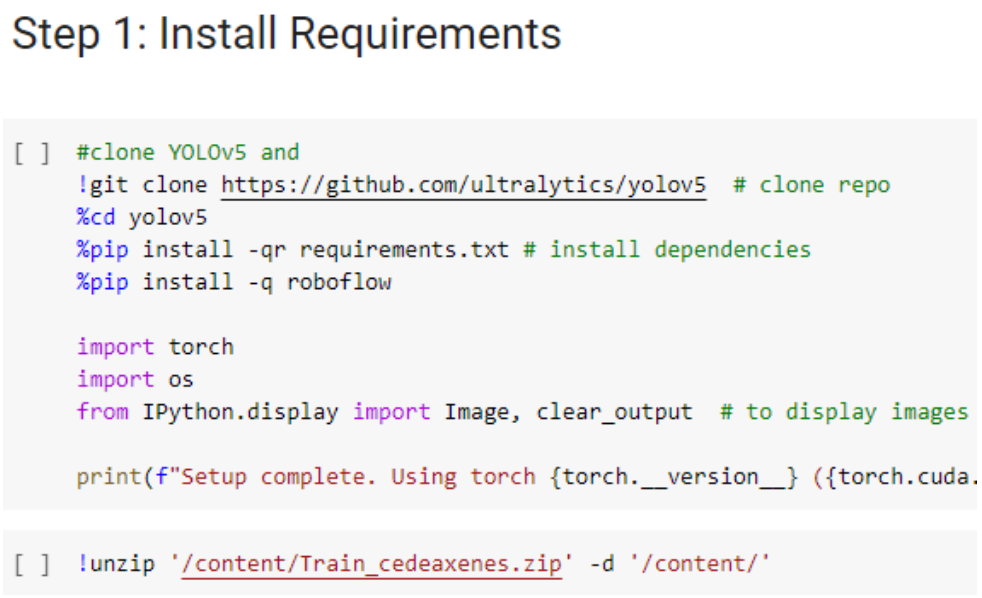
U ovome radu odabran je za korištenje Google Colab jer sam s njim bio najviše upoznat i preko njega je moguće besplatno (do neke mjere) koristiti Google-ove grafičke kartice dostupne preko Google Colab-a. Za korištenje YOLOv5 preko drugih izvora morala bi se koristiti vlastita grafička kartica, što je svakako kompleksniji pothvat s obzirom na to da bi bilo potrebno instalirati sve potrebne knjižnice i znati programirati korištenje grafičke kartice.

Slika 96. YOLOv5 [15]

8.2.1. Korištenje YOLOv5 algoritma na Google Colab-u

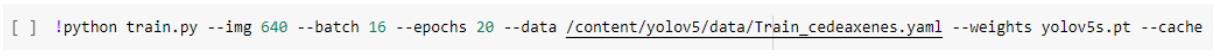
Korištenje YOLOv5 na Google Colab je relativno brz proces:

1. Otvori se YOLOv5 Google Colab sheet sYOLO github-a
2. Pokrene se prva ćelija kako bi se sve konfiguriralo

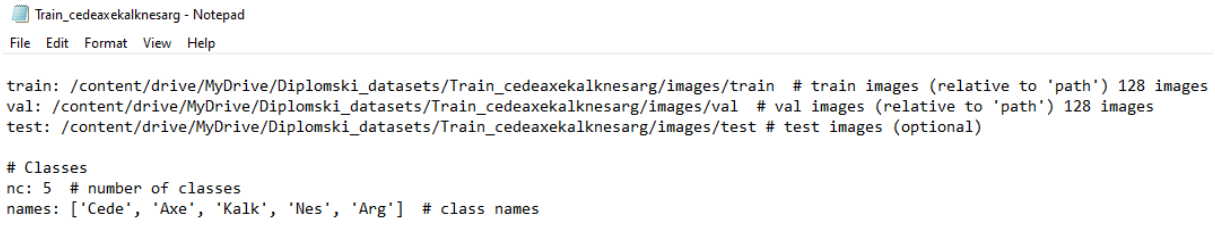


Slika 97. YOLOv5 Google Colab početak

3. Uplouda se .zip oblik set slika
4. U drugoj ćeliji (vidljivo na slici iznad) je naredba za unzip ove .zip datoteke kada se skroz upload-ala
5. Postaviti parametre: veličina slike, veličina *batcha*, broj epoha, .yaml datoteka na kojoj su upute gdje da neuronska mreža pronađe slike. *Batch* je broj koliko slika će neuronska mreža uzimati po iteraciji. Epoha je broj koliko će puta neuronska mreža proći kroz cijeli set slika. Zatim pokrenuti trening klikom na pokretanje ćelije gdje su se postavili parametri



Slika 98. YOLOv5 početak treninga



Slika 99. YOLOv5 datoteka s klasama i putanjama

U .yaml datoteci su sadržani točni putevi gdje se nalaze slike i oznake, te je naveden broj klasa odnosno broj predmeta koji se uči mrežu. Klase moraju ići redom kako su brojevi navedeni u tekstualnim datotekama svakog predmeta što bi znači da će Cedevita u svakoj

svojoj .txt datoteci prvi broj imati 0 (broj klase) i zatim 4 broja koji označuju koordinate pravokutnika za označavanje predmeta.

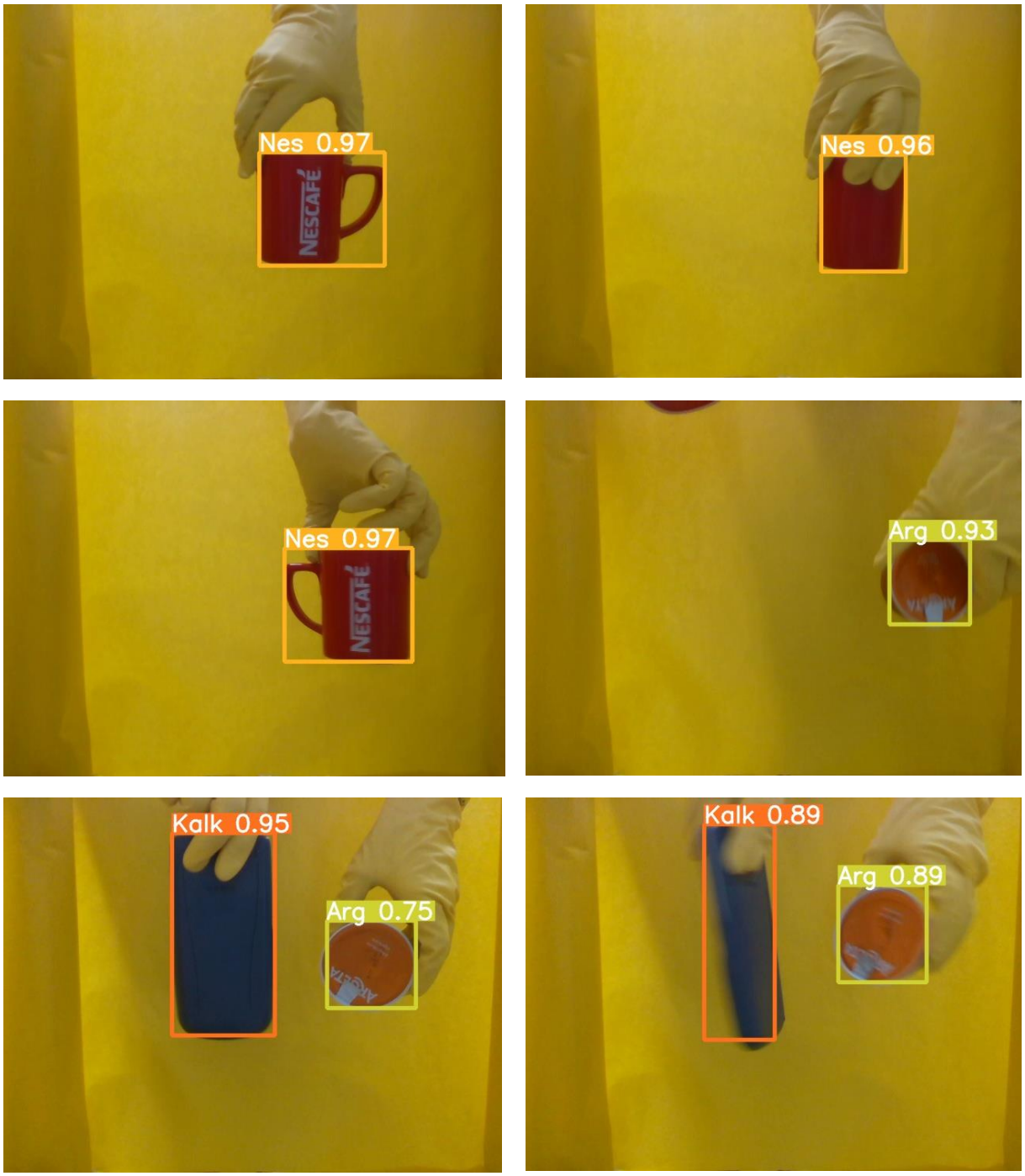
6. Isprobati treniranu neuronske mrežu na testnim slikama i videima. Tu je potrebno staviti veličinu testnih slika/videa, najnižu sigurnost detekciju i putanju do testnih slika/videa.

```
[ ]  !python detect.py --weights runs/train/exp/weights/best.pt --img 640 --conf 0.5 --source /content/novi_testni_speed.mp4
```

Slika 100. YOLOv5 testiranje

8.3. Evaluacija rezultata

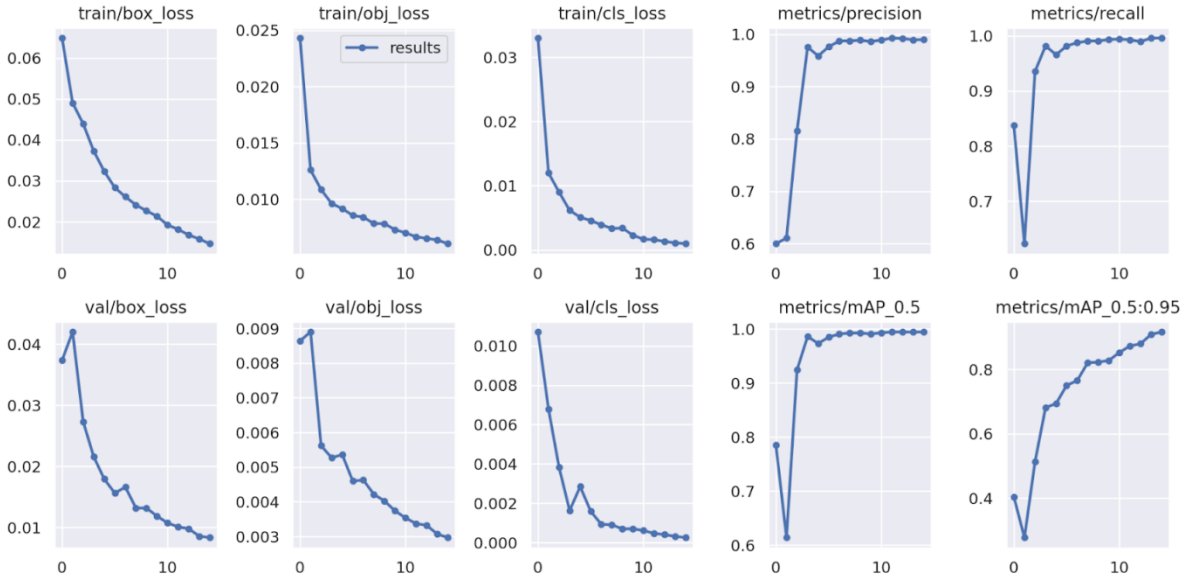
Neuronska mreža se učila za detekciju 5 različitih objekata, sve su to neki klasični proizvodi. Uzorak za učenje je imao nešto preko 700 slika svakog od predmeta. Rezultati provedeni na testnim slikama su prikazani ispod teksta. Brojevi iznad detektiranih objekata označuju koliko je mreža sigurna da se baš taj predmet nalazi u označenom pravokutniku.



Slika 101. Rezultati detekcije objekata na testnim slikama

Pod evaluaciju i analizu rezultata pripada i statistika koja je dostupna od strane *Tensorboard-a*. Analiza učenja mreže je prikazana na slici ispod, te se iz nje može iščitati kako gubitci padaju s brojem epoha a preciznost raste, a to znači da je mreža najvjerojatnije dobro naučila, iako pravi test je tek kada se vidi koliko dobro mreža detektira na testnim slikama/videima, što je također već pokazano da je u redu. Iz grafova ispod još se može zaključiti kako je za mrežu bilo dovoljno 10 epoha da se istrenira na zadovoljavajuću razinu. Broj epoha je nešto

što se ne može znati unaprijed koliko ih točno postaviti, ali se zato naknadnom analizom može utvrditi. Broj epoha je bitan zbog korištenja resursa i vremena treniranja mreže. S postavljenih 15 epoha trening ove mreže je, uz pomoć grafičke kartice, trajao 20-tak minuta, a smanjenjem broja epoha na 10, trening bi trajao 10-15 minuta.



Slika 102. Analiza treniranja neuronske mreže

9. ZAKLJUČAK

U ovom radu predstavljeno je rješenje za rekonstrukciju 3D geometrije proizvoda i za automatsko generiranje 2D seta slika za treniranje neuronskih mreža. Postav s tri kamere je simuliran, zatim stvarno realiziran i korišten za ovu primjenu. Za procesuiranje i obradu podataka koristili su se algoritmi izrađeni pomoću Open3D i OpenCV biblioteke. Sustav pokazuje zadovoljavajuće rezultate i kod 3D rekonstrukcije i kod generiranja 2D seta slika za treniranje neuronskih mreža unatoč svim mogućim greškama kod intrinzične kalibracije, stereo kalibracije, ICP registracije i greške pri vjernosti oblaka točaka i slika uzetih pomoću RealSense kamera. Dakle, kao prednost sustava može se navesti dokazana mogućnost 3D rekonstrukcije geometrije proizvoda, te generiranja seta slika spremnih za kvalitetno treniranje neuronskih mreža za kasniju detekciju. Ograničenja sustava ovise o uvjetima okoline koji moraju biti kontrolirani i konstantni, jer za segmentaciju pozadine iz slike i mala promjena svjetlosti iz okoline može igrati veliku ulogu, te dovesti do loših rezultata. Također, treba naglasiti kako uz bolje kamere i bolju kontrolu svjetla, sustav ima mjesta za napredak u vidu promjene papira korištenog za pozadinu u neku drugu boju koja se rjeđe koristi na proizvodima, zatim je moguća (i poželjna) primjena robota za držanje i rotaciju predmeta tijekom slikanja. Sustav je pogodan za daljnje nadogradnje i modifikacije korištenih algoritama obrade i upravljanja programom, gdje zasigurno ostaje prostora za razvoj u budućnosti.

LITERATURA

[1] Collins, R., Predavanja iz kolegija Introduction to Computer Vision, Penn State University, Pennsylvania, 2007.

[2] Pasivni i aktivni stereo sustavi: https://www.e-consystems.com/blog/camera/technology/what-is-a-stereo-vision-camera-2/, Pristupljeno: 11.lipnja 2022.

[3] Intel RealSense Camera D400 Series Product Family Datasheet, 2020.

[4] Način okidanja kamere: https://www.premiumbeat.com/blog/know-the-basics-of-global-shutter-vs-rolling-shutter/, Pristupljeno: 11. lipnja 2022.

[5] Anders Grunnet –Jepsen, Paul Winer, Aki Takagi, John Sweetser, Kevin Zhao, Tri Khuong, Dan Nie, John Woodfill – Multi-Camera configurations – D400 Series Stereo Cameras. Dostupno online: https://dev.intelrealsense.com/docs/multiple-depth-cameras-configuration

[6] Teorija kalibracije kamere: https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html, Pristupljeno: 18.lipnja 2022.

[7] Stereo Camera Calibratior: https://www.mathworks.com/help/vision/ug/using-the-stereo-camera-calibrator-app.html, Pristupljeno: 22.lipnja 2022.

[8] Teorija stereo kalibracije kamera: https://sourishghosh.com/2016/stereo-calibration-cpp-opencv/, Pristupljeno: 18.lipnja 2022.

[9] Struktura oblaka točaka i njegovi formati: https://support.zivid.com/latest/reference-articles/point-cloud-structure-and-output-formats.html, Pristupljeno: 23.lipnja 2022.

[10] Point Cloud Library (PCL): https://medium.com/@prarthana.uz8d.bataju/point-cloud-library-pcl-93907726c723, Pristupljeno: 23.lipnja 2022.

[11] Rezolucije pojašnjenje: https://www.viewsonic.com/library/tech/monitor-resolution-aspect-ratio/, Pristupljeno: 23.lipnja 2022.

[12] Praktični vodič za detekciju objekata pomoću YOLOv5 algoritma: https://towardsdatascience.com/the-practical-guide-for-object-detection-with-yolov5-algorithm-74c04aac4843, Pristupljeno: 23.lipnja 2022.

[13] Konvolucijski proces: https://www.researchgate.net/figure/A-convolution-process-with-a-filter-size-of-3-A-3_fig3_338979273, Pristupljeno: 23.lipnja 2022.

[14] Machine Learning Foundations: Ep #3 – Convolutions and pooling: https://www.youtube.com/watch?v=PCgLmzkRM38, Pristupljeno: 23.lipnja 2022.

[15] YOLOv5 github: https://github.com/ultralytics/yolov5, Pristupljeno: 23.lipnja 2022.

PRILOZI

I. CD-R disk