Razvoj programske podrške za robotski 3D strojni vid

Čutura, Marko

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentori:

Doc. dr. sc. Marko Švaco,
Dr. sc. Filip Šuligoj

Student:

Marko Čutura

Zagreb, godina. 2022

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

Zahvaljujem se svojem mentoru doc.dr.sc. Marku Švaci i komentoru dr.sc. Filipu Šuligoju na danoj pomoći i savjetima prilikom izrade rada.

Najviše se zahvaljujem ocu Blažanu i majci Mariji na podršci tokom studiranja.

Marko Čutura

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: 602-04/22-6/1	
Ur. broj: 15-1703-22-	

DIPLOMSKI ZADATAK

Student: **MARKO ČUTURA** Mat. br.: 0035212237

Naslov rada na hrvatskom jeziku: **Razvoj programske podrške za robotski 3D strojni vid**

Naslov rada na engleskom jeziku: **Development of a software framework for robotic 3D machine vision**

Opis zadatka:

Primjena industrijskih robota sve je češća u nestrukturiranoj radnoj okolini. U industrijskim pogonima predmeti rada s kojima roboti trebaju rukovati mogu biti slobodno raspoređeni u nizu, ravnini ili prostoru. Predmeti slobodno raspoređeni u prostoru predstavljaju najkompleksniju razinu problema za koju je često nužan razvoj ili implementacija naprednih algoritama strojnog vida s ciljem lokalizacije i registracije predmeta u svih šest dimenzija (tri prostorne translacije i tri rotacije).
U ovom radu potrebno je upoznati se s 3D vizijskim sustavom Zivid One te izvršiti obradu i vizualizaciju oblaka točaka koristeći knjižnicu otvorenog koda za obradu oblaka točaka PCL (eng. „Point cloud library"). Također potrebno je upoznati se s radom robotskog sustava Motoman CSDA10F, načinima programiranja s naglaskom na Motoplus programski dodatak. MotoPlus je knjižnica za razvoj softvera za specijalizirane aplikacije za upravljanje Motoman robotima te komunikaciju s vanjskim uređajima koristeći TCP/IP komunikacijski protokol.

U sklopu diplomskog rada potrebno je napraviti i sljedeće:
- Kalibrirati koordinatni sustav vizijskog sustava i robota.
- Proučiti postojeće algoritme za rad s oblacima točaka.
- Primijeniti algoritme za filtriranje šumova.
- Primijeniti algoritme prepoznavanje prostornih značajki i registraciju različitih 3D snimaka koristeći softverski paket PCL.
- Riješiti problem izuzimanja predmeta iz kutije (eng. „bin picking") koristeći 3D vizijski sustav i robotsku ruku.

Eksperimentalnu validaciju postava za rukovanje predmetima rada u nestrukturiranoj radnoj okolini potrebno je napraviti na dostupnom Motoman CSDA10F robotu u Laboratoriju za računalnu inteligenciju u CRTA-i.

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

Zadatak zadan:	Rok predaje rada:	Predviđeni datum obrane:
3. ožujka 2022.	5. svibnja 2022.	9. svibnja do 13. svibnja 2022.

Zadatak zadao:	Komentor:	Predsjednica Povjerenstva:
doc. dr. sc. Marko Švaco	Filip Šuligoj	prof. dr. sc. Biserka Runje

SADRŽAJ

POPIS SLIKA

POPIS OZNAKA

Oznaka	Jedinica	Opis
n		Članovi matrice homogenih transformacija
b, c		Konstantne vrijednosti
R		Matrica rotacija
R_{ij}		Članovi matrice rotacija
α	°	Kut oko osi x
β	°	Kut oko osi y
γ	°	Kut oko osi z
d	mm	Udaljenost između dvije točke u oblaku točaka
p		Točka u oblaku točaka
P		Skup točaka
d_m	mm	Maksimalna udaljenost
k		Broj susjednih točaka
F		Vektor deskriptora
Γ		Mjera sličnosti između dvije točke
μ_d		Prosječna srednja udaljenost
σ_d		Standardna devijacija
d_{th}	mm	Maksimalan dozvoljen prag udaljenosti kod euklidske klasterizacije
R_x, R_y,R_z	°	Eulerovi kutevi oko vrha alata
R_e	°	Pozicija vanjske osi robota
O		Klaster, skup točaka
d_t	mm	Maksimalna dozvoljena udaljenost *inliera* od ravnine
H		Matrica homogenih transformacija
H_{OBJ}^{ROB}		Matrica transformacija od baznog koordinatnog sustava robota do kalibracijske ploče
H_{OBJ}^{CAM}		Matrica transformacija od ishodišta kamere do objekta traženja
H_{EE}^{OBJ}		Matrica transformacija od kalibracijske ploče to prirubnice robota
H_{CAM}^{ROB}		Matrica transformacija od baznog koordinatnog sustava robota do ishodišta kamere
H_{Init}		Matrica transformacija dobivena Hough-ovim algoritmom klasterizacije
H_{ICP}		Matrica transformacija dobivena ICP algoritmom
H_{NEWOBJ}^{ROB}		Matrica transformacija od baznog koordinatnog sustava robota do pronađenog komada na sceni
H_{NEWOBJ}^{REFOBJ}		Matrica transformacija od referentnog objekta do pronađenog objekta na sceni

H_{REFOBJ}^{CAM}	Matrica transformacija od ishodišta kamere do referentnog objekta
H_{REFOBJ}^{ROB}	Matrica transformacija od baznog koordinatnog sustava robota do referentnog objekta

SAŽETAK

U ovom radu izrađen je sustav za lokalizaciju predmeta u nestrukturiranoj okolini. Implementiran je sustav koji se sastoji od Zivid 3D stereo kamere koju se može koristiti kroz Zivid Studio. U svrhu izrade sustava Zivid kamera implementirana je za *online* primjenu pomoću njezine C++ biblioteke koja dolazi od strane proizvođača. Uz kameru implementiran je algoritam za lokalizaciju baziran na prepoznavanju pomoću C++ PCL(Point Cloud Library) biblioteke. Korišteni su koncepti modela i scene, gdje je model objekt poznate pozicije i orijentacije, a scena predstavlja nestrukturiranu okolinu na kojoj se traži objekt ili komad od interesa, pomoću algoritma za lokalizaciju nalazi se objekt na sceni pomoću već postojećeg modela. Također, u radu je implementiran robotski manipulator Yaskawa CSDA10F uparen sa kontrolerom FS100 koji izvršava pozicioniranje TCP(*Tool Center Point*) alata u odnosu na lokalizirani objekt čime se potvrđuje uspješnost lokalizacije. Da bi se robot mogao implementirati napravljena je *Hand Eye* kalibracija robota sa Zivid kamerom, koja povezuje bazni koordinatni sustav robota sa ishodištem kamere. Napravljena je C aplikacija u Motoplus biblioteci koja omogućava sljedeće opcije potrebne za funkcioniranje rada: komunikaciju robota sa računalom pomoću *Socket* komunikacije, funkcije za transformaciju rezultata algoritama lokalizacije u pozicijsku varijablu robota i funkcije za manipulaciju ostalim varijablama i signalima robota. Da bi se prikazala praktična primjena navedenog rada, izrađena je *Bin Picking* aplikacija u kojoj robot izuzima predmet pronađen pomoću sustava za lokalizaciju i odlaže ga na unaprijed predviđeno mjesto. Za izuzimanje i manipulaciju predmetom konstruirana je i izrađena vakuumska prihvatnica.

Ključne riječi: stereo kamera, matrica homogenih transformacija, Yaskawa, CSDA10F, FS100, Motoplus, lokalizacija, prepoznavanje, C, C++, Point Cloud Library, Zivid, *Hand Eye* kalibracija, *Bin Picking*, vakuumska prihvatnica.

SUMMARY

In this thesis a system for localization of pieces in unstructured environment was developed. A system is implemented that consists of Zivid 3D stereo camera which can be used through Zivid Studio. For the purpose of building this system Zivid camera was implemented in an online fashion, using its own C++ Library supplied by the manufacturer. Alongside the camera an algorithm for localization based on recognition was implemented using the C++ PCL(Point Cloud Library) library. Concepts of model and scene were used, where the model stands for an object of known position and scene stands for an unstructured environment, using the localization algorithm it is possible to find an object on scene using an already known model of object. Also, in this thesis a robot manipulator Yaskawa CSDA10F was implemented paired up with the controller FS100, which executes the positioning of its TCP(Tool Center Point) in reference to the localized object, which confirms the success of localization. To implement the robot, a Hand Eye calibration procedure between the robot and the Zivid Camera was carried out, which connects the base coordinate system of the robot with the center of the camera. A C language application was created using the Motoplus library which allows the use of following options needed for system to function properly: communication of robot with the computer using the socket communication, functions for transformation of the results given by the localization algorithm to the position variables of the robot, and lastly, functions for manipulation with other robot variables and signals. To demonstrate the practical use of the aforementioned work, a Bin Picking application was developed in which a robot picks up the piece found using the system for localization and places it on a predetermined location. For the purpose of picking a vacuum gripper was designed and fabricated.

Key words: stereo camera, homogenous transformation matrix, Yaskawa, CSDA10F, FS100, Motoplus, localization, recognition, C, C++, Point Cloud Library, Zivid, Hand Eye calibration, Bin Picking, vacuum gripper.

1. UVOD

Jedan od izazova industrijske automatizacije danas je izuzimanje dijelova iz nestrukturirane ili polustrukturirane okoline. Radnja u kojoj robot izuzima dijelove iz neodređene okoline i raspoređuje ih prema predodređenom slijedu, u ovom slučaju to je *Bin Picking* aplikacija u kojoj je cilj da robot izuzme dijelove u nasumičnim pozicijama i orijentacijama nakon čega ih postavlja u predodređene pozicije, primjer toga je izuzimanje komada iz kutije i postavljanje na pokretnu traku. Izazovi su mnogi, određivanje robota sposobnog za manipulaciju željenim dijelom, mora imati dovoljne kinematske i dinamičke sposobnosti da bi postigao zadovoljavajuće ciklusno vrijeme, također konfiguracije njegovih osi moraju biti dovoljno fleksibilne da bi zahvatio proizvod u teško dostupnim pozicijama i orijentacijama. Potrebno je odabrati kameru s dovoljnom rezolucijom i sposobnostima kako bi se dobio precizan oblak točaka(engl. *Point Cloud*). Potrebno je izraditi aplikaciju kojom bi se izvršavalo prepoznavanje dijelova i dobivanje potrebnih informacija o poziciji i orijentaciji. Nakon dobivanja potrebnih informacija, potrebno je pomoću industrijske komunikacije podatke prebaciti na robota, koji izvršava obradu i prilagodbu na način da dobije poziciju objekta u svom baznom koordinatnom sustavu.

2. Osnovni koncepti

Za razumijevanje rada potrebno je razjasniti osnovne korištene koncepte, bez kojih nema smisla čitati rad. Jedan od njih je matrica homogenih transformacija koja određuje poziciju i orijentaciju predmeta u prostoru. Drugi je euklidska udaljenost koja predstavlja najkraću udaljenost između dvije točke u prostoru.

2.1. Matrice homogenih transformacija

Da bi se mogla odrediti pozicija i orijentacija objekta u 3D prostoru, potrebno je matematički opisati translacije i rotacije točaka i predmeta. Objekti u 3D prostoru opisani su svojim translacijama i rotacijama u odnosu na polazni koordinatni sustav. Navedene translacije i rotacije definiraju se matricama homogenih transformacija koje su dimenzija 4x4 za 3D prostor. Notacija korištena u ovom radu je:

$$H = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

Gdje je n vektor normale, o je vektor orijentacije, a je vektor djelovanja i p je vektor položaja. Treba naglasiti da matrice homogenih transformacija nisu komutativne tj.

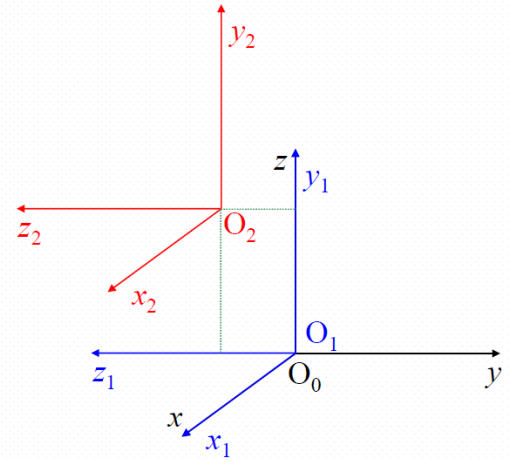
$$H_1^0 \cdot H_2^1 \neq H_2^1 \cdot H_1^0 \tag{2}$$

2.1.1. Relativne transformacije

Matrice relativnih transformacija definiraju transformaciju pozicije na sljedeći način:

$$H = H_1^0 \cdot H_2^1 \tag{3}$$

Što je prikazano na slici 1 ispod:

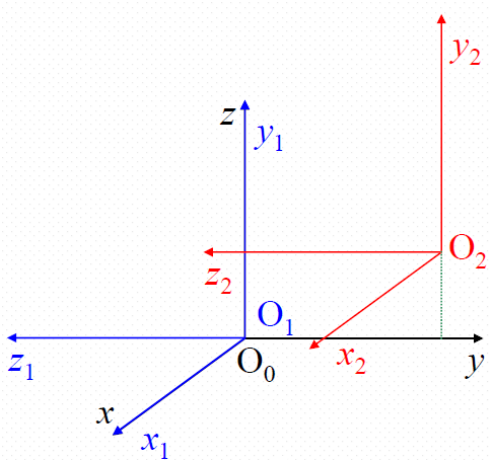


Slika 1. Djelovanje relativnih transformacija[1]

Matrice apsolutnih transformacija definiraju transformaciju pozicije na sljedeći način:

$$H = H_2^1 \cdot H_1^0 \tag{4}$$

Što je prikazano na slici 2 ispod:



Slika 2. Djelovanje apsolutnih transformacija[1]

Gdje u obadva slučaja matrice imaju isto značenje:

$$H_1^0 = Rot(x, 90°) \tag{5}$$

$$H_2^1 = Tran(0, b, c) \tag{6}$$

Gdje su b i c neke konstantne vrijednosti.

2.2. Eulerovi kutevi

U ovom radu koristit će se Eulerova z,y',x'' konvencija prikaza rotacija budući da manipulator koji se koristi u nastavku rada koristi tu konvenciju. Ako se iz jednadžbe (1) uzmu samo članovi za rotaciju dobiva se sljedeći izraz:

$$R = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} = \begin{bmatrix} n_x & o_x & a_x \\ n_y & o_y & a_y \\ n_z & o_z & a_z \end{bmatrix} \tag{7}$$

Gdje R predstavlja matricu rotacija.

Ako se definira rotacija oko osi X,Y i Z na način da α predstavlja rotaciju oko osi X, β rotaciju oko osi Y i γ za rotaciju oko osi z, pomoću matrice R mogu se dobiti vrijednosti kuteva α, β i γ. Prema izrazu (8) ispod može se vidjeti relacija između Eulerovih kutova i matrice rotacija R:

$$R = \begin{bmatrix} \cos(\alpha)\cos(\beta) & cos(\alpha)sin(\beta)sin(\gamma) - cos(\gamma)sin(\alpha) & sin(\gamma)sin(\alpha) + c(\gamma)c(\alpha)s(\beta) \\ \cos(\beta)\sin(\alpha) & cos(\alpha)cos(\gamma) + sin(\gamma)sin(\beta)sin(\alpha) & cos(\gamma)sin(\alpha)sin(\beta) - cos(\alpha)sin(\gamma) \\ -\sin(\beta) & cos(\beta)sin(\gamma) & cos(\beta)cos(\gamma) \end{bmatrix} \tag{8}$$

Dakle, matrica rotacija R definira kuteve oko osi. Treba naglasiti da se nakon primjenjivanja jedne rotacije nova rotacija primjenjuje na novonastalu konfiguraciju osi, tj. npr. nakon primjenjivanja zakreta γ oko osi Z, zakret β se primjenjuje na koordinatni sustav nastao rotacijom oko Z, isto tako vrijedi i za posljednu rotaciju α oko osi X.

2.3. Euklidska udaljenost

Euklidska udaljenost definira se kao udaljenost d između dvije točke $p_1(x_1, y_1, z_1)$ i $p_2(x_2, y_2, z_2)$ u 3D prostoru prema jednadžbi:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{9}$$

3. Vizijski sustav

3.1. Oblak točaka

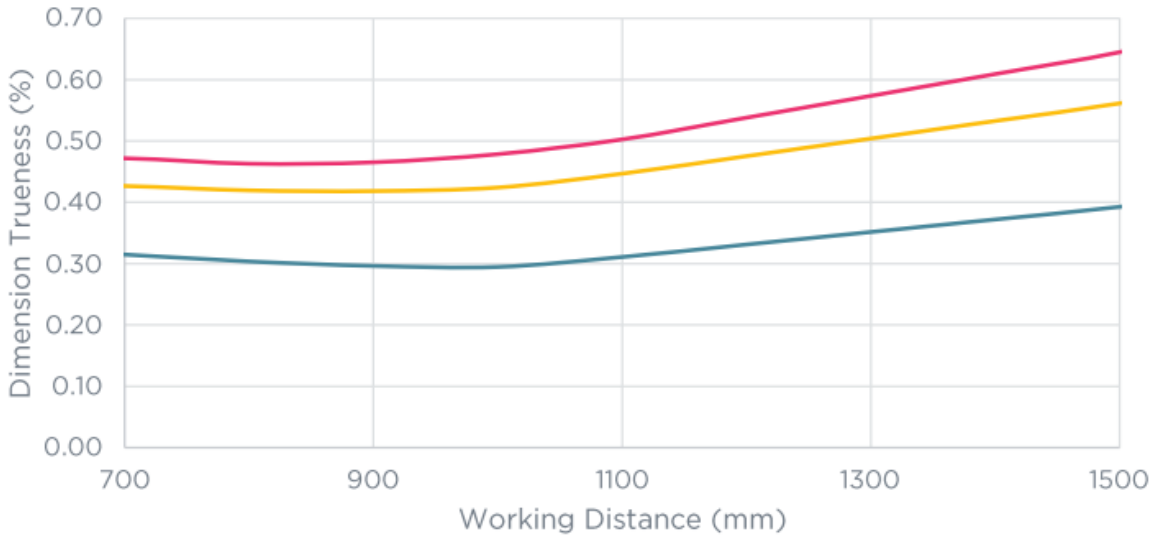
Oblak točaka ili eng. *Point Cloud* je format kojim bi se opisala 3D slika. Svaka točka u oblaku točaka opisana je svojom x, y i z koordinatom uz mogućnost dodavanja vrijednosti boje RGB. Koordinate x, y i z predstavljaju udaljenost točke od senzora kamere. U matematičkom smislu oblaci točaka predstavljaju višedimenzionalne matrice koje sadrže podatke o poziciji i boji točke. Prednost oblaka točaka je jednostavna manipulacija, dok je mana potreba za velikom količinom memorije i procesorske moći što čini izvođenje nekih geometrijskih operacija vrlo zahtjevnim. U većini slučajeva, različiti proizvođači mogu imati svoje formate oblaka točaka npr. Zivid ima .zdf dok PCL ima .pcd format. Algoritmi za konverziju su uglavnom dani od proizvođača 3D kamere.

3.2. Zivid One+ Medium

Zivid One+ Medium je stereo kamera sa 2.3 MPx senzorom(vidi sliku 3). Ima mogućnost slikanja 2D i 3D slika u boji, što pruža značajnu fleksibilnost pri opcijama kod integracije. Također omogućava dobivanje preciznih oblaka točaka kod sjajnih površina, što se dobiva robusnim i fleksibilnim algoritmima za potiskivanje takvih pojava, nešto što je značajan problem kod ostalih tehnologija [2]. Problemi s refleksijom suzbijaju se pomoću ART tehnologije(engl. *Artifact Reduction Technology*), koja uključuje *ContrastReduction* filter koji uklanja primarne izvore nastajanja pogrešaka. Zivid ima prednost visoke istinitosti slike. Istinitost slike uključuje greške nastale pri skaliranju(objekti se čine manji ili veći nego što jesu) i rotacijske te translacijske greške(objekti se prepoznaju u drugačijim pozicijama nego u kojima stvarno jesu), ovakvom problemu se može doskočiti korištenjem dovoljno fleksibilne prihvatnice, ali i dalje bi bilo jednostavnije korištenje precizne kamere, zato Zivid osigurava dimenzijsku točnost predmeta od 0.2% do 0.8% kroz cijelo vidno polje. Prednost Zivid-a je što ne koristi klasičnu konfiguraciju 3D kamere uz posebnu 2D kameru u boji kako bi dobila RGB informaciju. Zivid kamera ima potpuno integriran 3D sa RGB senzorom. Nije potrebna dodatna kalibracija RGB senzora sa XYZ podacima [3].

Slika 3. Zivid One+ Medium kamera [4]

Fokalna duljina iznosi 1000 mm. Optimalna radna udaljenost iznosi fleksibilnih 700 do 1500 mm, dok je maksimalni mogući raspon radne udaljenosti od 500 do 2000 mm. U ovom radu korištena je udaljenost od 1000 mm da bi se zadržala dobra prostorna rezolucija koja na 1000 mm iznosi 0.37 mm uz vidno polje od 702 x 432 mm [5][6]. Prikaz ovisnosti radne udaljenosti o dimenzijskoj točnosti vidljiv je na slici 4.

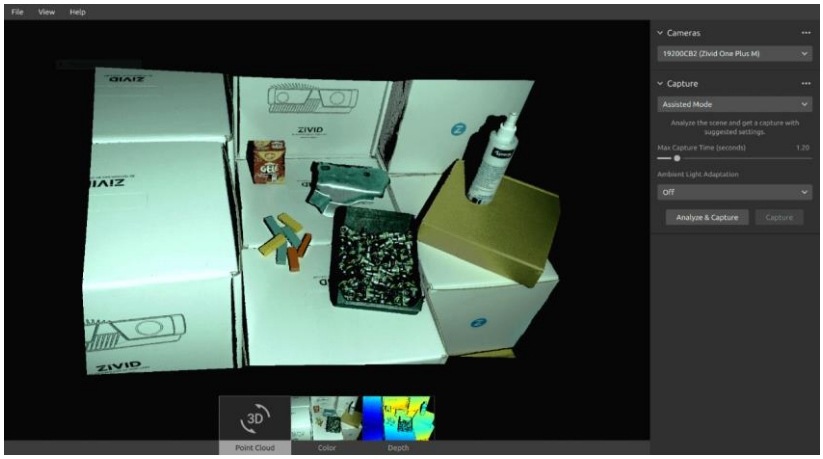


Slika 4. Dimenzijska točnost kamere u odnosu na radnu udaljenost [5]

Zbog svojeg visokog dinamičkog raspona kamera pokazuje dobre performanse kod tamnih i sjajnih metalnih površina. Uz sve prednost kamere postoje nepogodni slučajevi kao npr. slikanje prozirnih dijelova, zrcalne površine, slikanje objekata u pokretu i korištenje u primjenama gdje su potrebne točnosti ispod 0.03 mm [6].

3.3. Zivid Studio

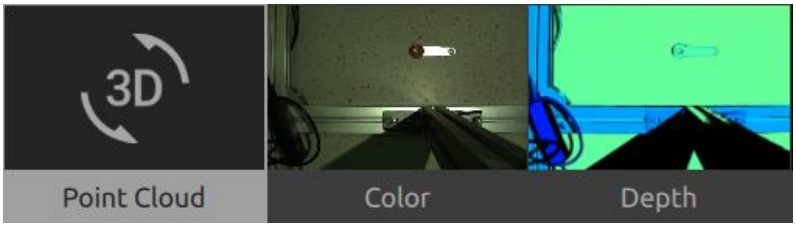
Zivid Studio je GUI posebno napravljen za interakciju za Zivid kamerama. Pomoću njega može se direktno upravljati sa kamerom, podesiti parametri kao: svjetlina, ekspozicija, pojačanje, filtracija buke, refleksije, ispravak distorzije i sl. Vrlo korisna opcija je korištenje tzv. *Assisted* načina slikanja. U ovakvom načinu rada, kamera uzima nekoliko slika tako da svaku prijašnju sliku analizira i podešava parametre za sljedeću tj. koristi iterativni postupak modifikacije parametara. Tako dobiveni parametri mogu se koristiti za programiranje kamere u Visual Studio programskom paketu gdje postoji poseban Zivid SDK. Dakle, pomoću Zivid Studio-a možemo dobiti početne parametre i generalni izgled oblaka točaka. Kao generalna preporuka, prije pokretanja sustava ne bi bilo loše izvesti *Assisted* seriju slikanja, čime bi se umjerila kamera u odnosu na vanjske utjecaje, nakon čega bi se ti parametri mogli unijeti pomoću posebne rutine u program. Kamera se na industrijsko računalo spaja pomoću oklopljenog USB kabela [6]. Prikaz GUI sučelja vidljiv je na slici 5.



Slika 5. Zivid Studio GUI sučelje [6]

Za pregledavanje dobivenih oblaka točaka moguće je koristiti tri načina gledanja. *Point Cloud* gdje je moguće vidjeti oblak u boji, te ga je moguće rotirati i translatirati po želji.

Sljedeći pogled je *Color* gdje je moguće vidjeti oblak točaka iz tlocrta s vrlo točno definiranim bojama u prostoru. Zadnji pogled je *Depth*, gdje je moguće pregledati dubinu točaka prelaženjem kursorom miša preko njih. Pregled pogleda vidljiv je na slici 6.



Slika 6. Zivid Studio, dostupni pogledi [6]

3.4. Zivid SDK

Zivid SDK je dodatak Visual Studio programskom paketu pomoću kojeg se može implementirati parametriranje i korištenje Zivid kamere u programskom okruženju C++ jezika. Moguće je spajanje na kameru, podešavanje parametara kao i u Zivid Studio-u, *online* uzimanje oblaka točaka, spremanje slika i konverzija slike u različite oblike npr. XYZRGB ili XYZ .pcd formate. Također je moguće simulirati rad kamere pomoću učitavanja već postojećih .zdf datoteka(.zdf je predodređeni format u koji Zivid sprema svoje slike). Nakon učitavanja .zdf datoteke moguće je napraviti konverziju u .pcd format, čime se omogućava daljnja obrada slike. Primjer koda u Visual Studio razvojnom okruženju vidljiv je na slici 7.

```
try
{
    Zivid::Application zivid;

    std::cout << "Connecting to camera" << std::endl;
    auto camera = zivid.connectCamera();

    std::cout << "Creating settings" << std::endl;
    const auto settings = Zivid::Settings{ Zivid::Settings::Acquisitions{
        Zivid::Settings::Acquisition{ Zivid::Settings::Acquisition::Aperture{ 5.66 } } } };

    std::cout << "Capturing frame" << std::endl;
    const auto frame = camera.capture(settings);

    std::cout << "Setting up visualization" << std::endl;
    Zivid::Visualization::Visualizer visualizer;

    std::cout << "Visualizing point cloud" << std::endl;
    visualizer.showMaximized();
    visualizer.show(frame);
    visualizer.resetToFit();

    std::cout << "Running visualizer. Blocking until window closes" << std::endl;
    visualizer.run();

    const auto pointCloud = frame.pointCloud();
    const auto data = pointCloud.copyData<Zivid::PointXYZColorRGBA>();

    std::cout << "Creating PCL point cloud structure" << std::endl;
    pcl::PointCloud<pcl::PointXYZRGB> pointCloudPCL;

    std::cout << "Filling in point cloud data" << std::endl;
    pointCloudPCL.width = pointCloud.width();
    pointCloudPCL.height = pointCloud.height();
    pointCloudPCL.is_dense = false;
    pointCloudPCL.points.resize(pointCloudPCL.width * pointCloudPCL.height);

    for(size_t i = 0; i < pointCloudPCL.points.size(); ++i)
    {
        pointCloudPCL.points[i].x = data(i).point.x; // NOLINT(cppcoreguidelines-pro-type-union-access)
        pointCloudPCL.points[i].y = data(i).point.y; // NOLINT(cppcoreguidelines-pro-type-union-access)
        pointCloudPCL.points[i].z = data(i).point.z; // NOLINT(cppcoreguidelines-pro-type-union-access)
        pointCloudPCL.points[i].r = data(i).color.r; // NOLINT(cppcoreguidelines-pro-type-union-access)
        pointCloudPCL.points[i].g = data(i).color.g; // NOLINT(cppcoreguidelines-pro-type-union-access)
        pointCloudPCL.points[i].b = data(i).color.b; // NOLINT(cppcoreguidelines-pro-type-union-access)
    }

    pcl::PointCloud<pcl::PointXYZRGB>::Ptr cloudPtr(new pcl::PointCloud<pcl::PointXYZRGB>);
    *cloudPtr = pointCloudPCL;
```

Slika 7. Primjer koda za dobivanje slike s kamere i konverziju u .pcd format [7]

Također postoji niz ostalih primjera koda kao npr. za konverziju koordinatnih sustava, *Hand Eye* kalibraciju, smanjivanje rezolucije itd. Takvi primjeri dani od strane Zivid-a služe kao početna točka za implementaciju kamere u konačnu aplikaciju.

3.5. Hand Eye kalibracija kamere

Da bi podatci s 3D kamere bili primjenjivi na robotu, potrebno je izvršiti tzv. *Hand Eye* kalibraciju. *Hand Eye* kalibracija predstavlja poveznicu između koordinatnog sustava robota i koordinatnog sustava kamere. Kamera svoje točke definira u koordinatnom sustavu kamere, koji je definiran od proizvođača. Dok robot svoje pozicijske varijable ima definirane u svom baznom koordinatnom sustavu. Treba razlikovati dvije vrste kalibracije kod 3D stereo kamera, *Eye in Hand* i *Eye to Hand*.

3.5.1. Eye in Hand

Eye in Hand princip podrazumijeva da robot drži kameru , prednost ove metode je dobivanje oblaka točaka iz različitih pozicija robota, čime se dobiva značajna fleksibilnost u radu. Mana ove metode je potreba za implementacijom mehanizma za sprječavanje kolizije kamere s okolinom, dodatno opterećenje robota zbog mase kamere, nemogućnost uzimanja slika dok robot odlaže pronađeni dio i potrebno smanjenje brzine robota zbog sigurnosti kamere [8], prikaz ovakvog načina rada vidljiv je na slici 8.

Slika 8. *Eye in Hand* princip rada [8]

3.5.2. Eye to Hand

Eye to Hand način rada podrazumijeva da kamera ima statičnu poziciju u prostoru kroz koju promatra okoliš robota. Prednost ove metode je veća udaljenost robota od kamere i manja opasnost od kolizije, kamera može izvršiti akviziciju oblaka točaka dok robot obavlja neku drugu radnju npr. odlaganje zahvaćenog komada što može dovesti do nižeg ciklusnog vremena.

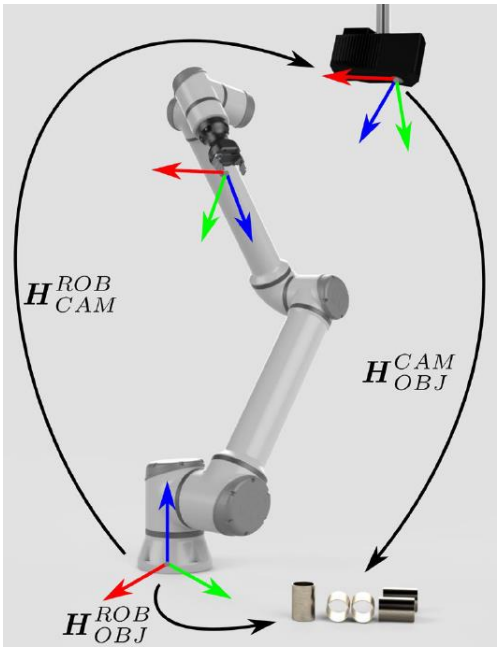
Mana ovog načina rada može biti okluzija predmeta od strane robota npr. robotska ruka se u trenutku slikanja nalazi između kamere i objekta. Druga mana je manja fleksibilnost kamere budući da se nalazi u statičnom položaju, također je potrebno izraditi platformu na koju bi kamera bila postavljena , što dovodi do dodatnog troška [8]. Unatoč manama ove metode ona je odabrana kao način rada kamere u ovom radu, prikaz ovakvog načina rada vidljiv je na slici 9.

Slika 9. Eye to Hand princip rada [8]

3.5.3. Zadatak Hand Eye kalibracije

Prilikom dohvaćanja nekog objekta, robot mora znati njegovu poziciju u odnosu na svoj bazni koordinatni sustav, uz poznavanje geometrije robota, pozicije i orijentacije izvršnog elementa(prihvatnice) može se izvršiti pozicioniranje prihvatnice robota do željenog objekta. Pri uobičajenim industrijskim aplikacijama moguće je osigurati poziciju objekta pomoću raznih sklopova, dodavača i steznih naprava. Povratna informacija o dolasku objekta u poziciju mogla bi se dati pomoću senzora. Kada bi senzor poslao povratnu informaciju, robot bi otišao u unaprijed spremljenu poziciju objekta od interesa i izuzeo predmet. U aplikacijama gdje pozicija objekta rada nije strogo definirana ne postoji ta mogućnost. Koristi se kamera i algoritmi lokalizacije da bi se dobila informacija o poziciji, nakon čega se vrši kompenzacija putanje robota. Problem nastaje kada se želi prenesti informacija sa kamere na robota. Koordinate oblaka točaka dobivaju se u koordinatnom sustavu kamere, kod Zivid-a taj se sustav nalazi na sredini unutarnje 2D kamere. Da bi ti podaci bili primjenjivi na realnom robotu, potrebno je izvršiti transformaciju koordinatnog sustava kamere u koordinatni sustav robota.

Za to služi *Hand Eye* kalibracija [8], odnos koordinatnih sustava kamere i robota prikazan je na slici 10.



Slika 10. Svrha Hand Eye kalibracije [8]

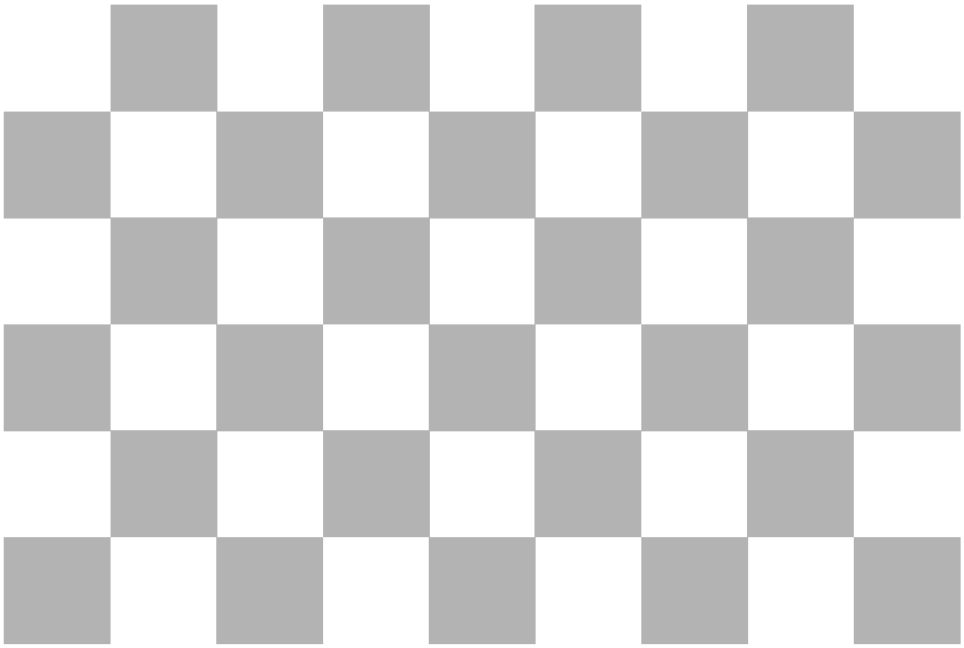
Prema slici 10 objekt manipulacije u koordinatnom sustavu robota može se dobiti pomoću sljedeće jednadžbe:

$$H_{OBJ}^{ROB} = H_{CAM}^{ROB} \cdot H_{OBJ}^{CAM} \tag{10}$$

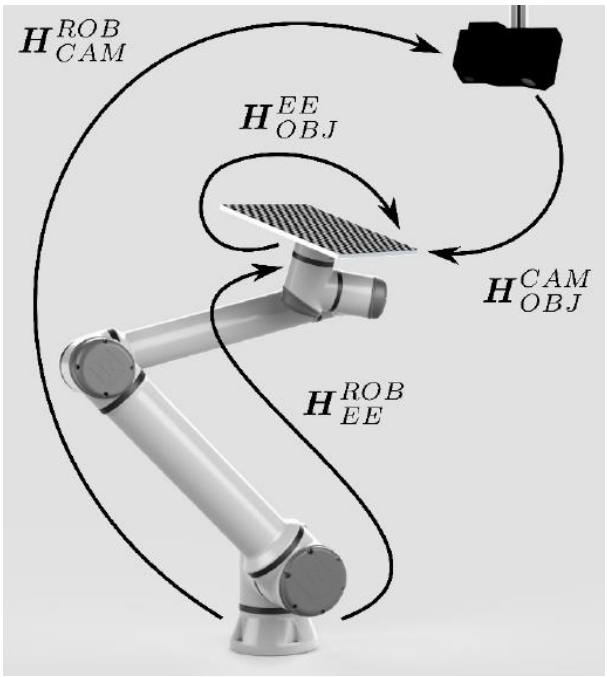
Gdje je H_{CAM}^{ROB} matrica transformacija koju dobivamo *Hand Eye* kalibracijom, ta matrica služi da bi se povezao koordinatni sustav robota s koordinatnim sustavom kamere.

3.5.4. Realizacija Hand Eye kalibracije

Za realizaciju kalibracije koristi se ploča sa crno bijelim kvadratima(vidi sliku 11). Ta ploča mora biti poznate geometrije. Slikanjem te ploče može se pomoću kamere procijeniti njena pozicija u odnosu na kameru. Istovremeno poznate su koordinate prirubnice robota u koordinatnom sustavu robota. Ove dvije činjenice povezuju se tako da se kalibracijska ploča pričvrsti na prirubnicu robota. Sada je situacija takva da postoji poznata transformacija između kamere i kalibracijske ploče H_{OBJ}^{CAM},poznata transformacija između baze i prirubnice robota H_{EE}^{ROB} i nepoznata transformacija između prirubnice i kalibracijske ploče H_{EE}^{OBJ} neovisna o poziciji robota. Pomoću poznavanja ovih podataka moguće je primijeniti jednu od optimizacijskih tehnika npr. Tsai-eva metoda, čime bi se dobila željena matrica transformacija H_{CAM}^{ROB} [8], opisani postupak vizualiziran je na slici 12 .

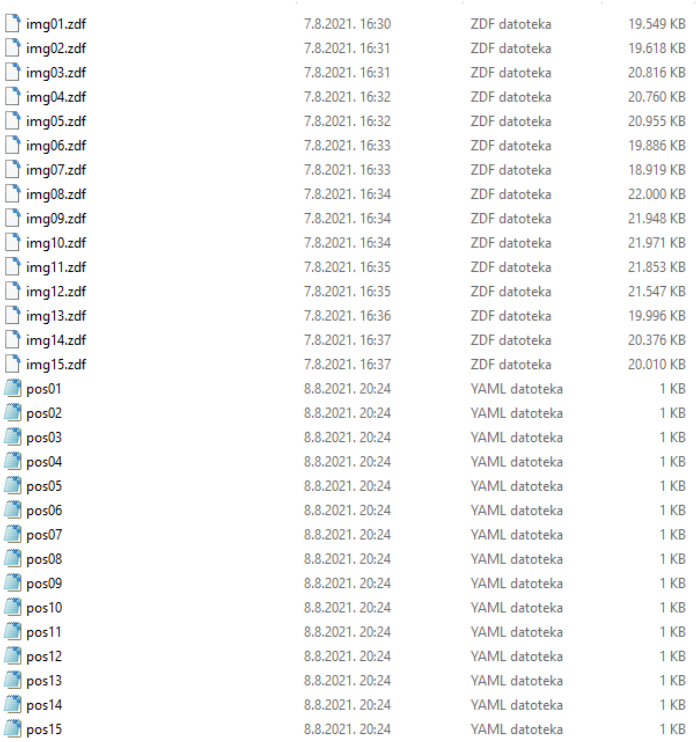


Slika 11. Kalibracijska podloga za Zivid One+ Medium kameru(9x6-20mm) [8]



Slika 12. Postupak *Hand Eye* kalibracije Zivid kamere [8]

Da bi se primijenile optimizacijske tehnike mora postojati dovoljno velik raspon uzoraka. Trebalo bi uzeti između 10 i 20 oblaka točaka u kojima robot drži kalibracijsku ploču, također bitno je da oblaci točaka budu što raznovrsniji u smislu da robot bude u raznovrsnim pozicijama zglobova. Pozicije robota su unaprijed poznate i spremljene u .yaml format. Spremljene .yaml datoteke moraju biti poredane u parovima s pripadajućim oblacima točaka kao na slici 13.



img01.zdf	7.8.2021. 16:30	ZDF datoteka	19.549 KB
img02.zdf	7.8.2021. 16:31	ZDF datoteka	19.618 KB
img03.zdf	7.8.2021. 16:31	ZDF datoteka	20.816 KB
img04.zdf	7.8.2021. 16:32	ZDF datoteka	20.760 KB
img05.zdf	7.8.2021. 16:32	ZDF datoteka	20.955 KB
img06.zdf	7.8.2021. 16:33	ZDF datoteka	19.886 KB
img07.zdf	7.8.2021. 16:33	ZDF datoteka	18.919 KB
img08.zdf	7.8.2021. 16:34	ZDF datoteka	22.000 KB
img09.zdf	7.8.2021. 16:34	ZDF datoteka	21.948 KB
img10.zdf	7.8.2021. 16:34	ZDF datoteka	21.971 KB
img11.zdf	7.8.2021. 16:35	ZDF datoteka	21.853 KB
img12.zdf	7.8.2021. 16:35	ZDF datoteka	21.547 KB
img13.zdf	7.8.2021. 16:36	ZDF datoteka	19.996 KB
img14.zdf	7.8.2021. 16:37	ZDF datoteka	20.376 KB
img15.zdf	7.8.2021. 16:37	ZDF datoteka	20.010 KB
pos01	8.8.2021. 20:24	YAML datoteka	1 KB
pos02	8.8.2021. 20:24	YAML datoteka	1 KB
pos03	8.8.2021. 20:24	YAML datoteka	1 KB
pos04	8.8.2021. 20:24	YAML datoteka	1 KB
pos05	8.8.2021. 20:24	YAML datoteka	1 KB
pos06	8.8.2021. 20:24	YAML datoteka	1 KB
pos07	8.8.2021. 20:24	YAML datoteka	1 KB
pos08	8.8.2021. 20:24	YAML datoteka	1 KB
pos09	8.8.2021. 20:24	YAML datoteka	1 KB
pos10	8.8.2021. 20:24	YAML datoteka	1 KB
pos11	8.8.2021. 20:24	YAML datoteka	1 KB
pos12	8.8.2021. 20:24	YAML datoteka	1 KB
pos13	8.8.2021. 20:24	YAML datoteka	1 KB
pos14	8.8.2021. 20:24	YAML datoteka	1 KB
pos15	8.8.2021. 20:24	YAML datoteka	1 KB

Slika 13. Raspored podataka za *Hand Eye* kalibraciju

Pojedinačna informacija o poziciji ima formu u obliku matrice homogenih transformacija, mora biti spremljena na sljedeći način u .yaml datoteci(vidi sliku 14).

```
%YAML:1.0
---
PoseState: !!opencv-matrix
    rows: 4
    cols: 4
    dt: d
    data:
        [
            0.124878,
            0.942448,
            0.310158,
            564.929000,
            -0.963063,
            0.190306,
            -0.190508,
            -7.363000,
            -0.238569,
            -0.274911,
            0.931402,
            -268.235000,
            0.000000,
            0.000000,
            0.000000,
            1.000000
        ]
```

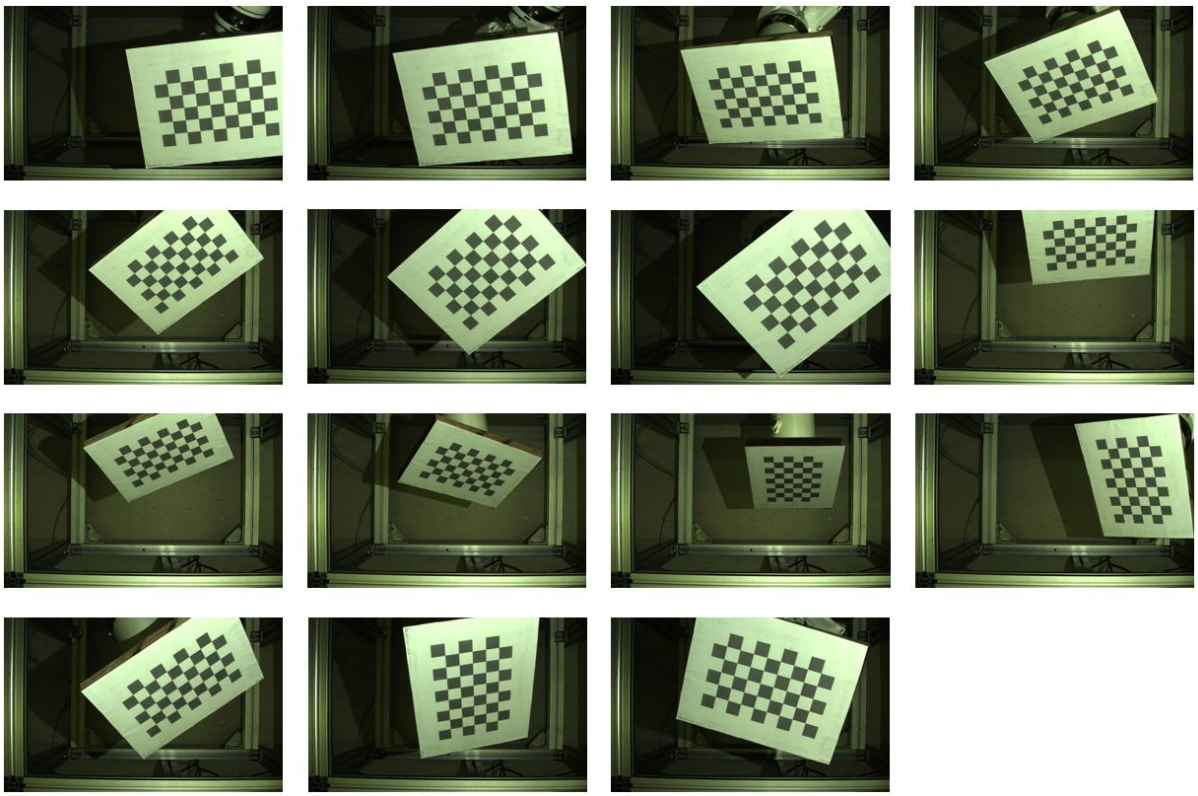
Slika 14. Prikaz pozicije robota za *Hand Eye* kalibraciju

Nakon što se uzme 10 do 20 uzoraka treba implementirat C++ *Snippet* vezan za *Hand Eye* kalibraciju. Prema ovom djelu koda može se vidjeti proces kalibracije(vidi sliku 15). Za početak program se spaja sa kamerom pomoću naredbe *Zivid.connectCamera()* . Nakon spajanja stvara se vektor *input* u koji se u paru spremaju pojedinačni oblak točaka i pozicija robota u tom trenutku. U *do* petlji pomoću *switch* naredbe može se odabrati između dvije opcije *cmdAddPose* i *cmdCalibrate* . Pomoću *cmdAddPose* pokreće se program za izvršavanje prethodno navedene radnje(dobivanje oblaka točaka i upisivanje pozicije robota). Kada se procijeni da je prisutan dovoljan broj uzoraka odabire se naredba *cmdCalibrate* pomoću koje se izvršava kalibracija. Ako je kalibracija uspješna program će vratiti 4x4 matricu homogenih transformacija između baznog koordinatnog sustava robota i ishodišta kamere [8]. Serija od 15 oblaka točaka korištenih u ovom radu vidljiva je na slici 16. Treba naglasiti da su mogući problemi prilikom izvođenja algoritma za kalibraciju u C++ okruženju, ako je to slučaj, potrebno je kalibraciju izvesti u Python okruženju(primjer korišten u ovom radu dan je u prilogu). Postupak je gotovo isti u oba slučaja.

Za uzimanje oblaka točaka korišten je *Assisted Capture* mod rada. Ako bi bilo eventualnih problema sa ovom metodom, moguće je koristiti potpuno ručno podešavanje parametara kamere kao npr. *iris*, vrijeme ekspozicije, svjetlina projektora i sl. Cilj podešavanja parametara je postaviti vrijednost piksela u područje histograma između 64 i 256 [8]. Više o ovom načinu kalibracije može se saznati iz literature [8]. Također, ako korisnik kamere nije dovoljno vješt u korištenju Visual Studio programskog paketa, C++ ili Python jezika, može koristiti postojeći *command prompt* alat Zivid CLI tool, alat posebno napravljen za *Hand Eye* kalibraciju pomoću kojeg se izbjegava potreba za programiranjem.

```
try
{
    Zivid::Application zivid;

    std::cout << "Connecting to camera" << std::endl;
    auto camera{ zivid.connectCamera() };

    size_t currPoseId{ 0 };
    bool calibrate{ false };
    std::vector<Zivid::Calibration::HandEyeInput> input;
    do
    {
        switch(enterCommand())
        {
            case CommandType::cmdAddPose:
            {
                try
                {
                    const auto robotPose = enterRobotPose(currPoseId);

                    const auto frame = assistedCapture(camera);

                    std::cout << "Detecting checkerboard in point cloud" << std::endl;
                    const auto result = Zivid::Calibration::detectFeaturePoints(frame.pointCloud());
                    if(result)
                    {
                        std::cout << "OK" << std::endl;
                        input.emplace_back(Zivid::Calibration::HandEyeInput{ robotPose, result });
                        currPoseId++;
                    }
                    else
                    {
                        std::cout << "FAILED" << std::endl;
                    }
                }
                catch(const std::exception &e)
                {
                    std::cout << "Error: " << Zivid::toString(e) << std::endl;
                    continue;
                }
                break;
            }
            case CommandType::cmdCalibrate:
            {
                calibrate = true;
                break;
            }
            case CommandType::cmdUnknown:
            {
                std::cout << "Error: Unknown command" << std::endl;
                break;
            }
        }
    } while(!calibrate);

    const auto calibrationResult{ performCalibration(input) };

    if(calibrationResult.valid())
    {
        std::cout << "Hand-eye calibration OK\n"
                  << "Result:\n"
                  << calibrationResult << std::endl;
```

Slika 15. *Snippet* za Hand Eye kalibraciju [7]



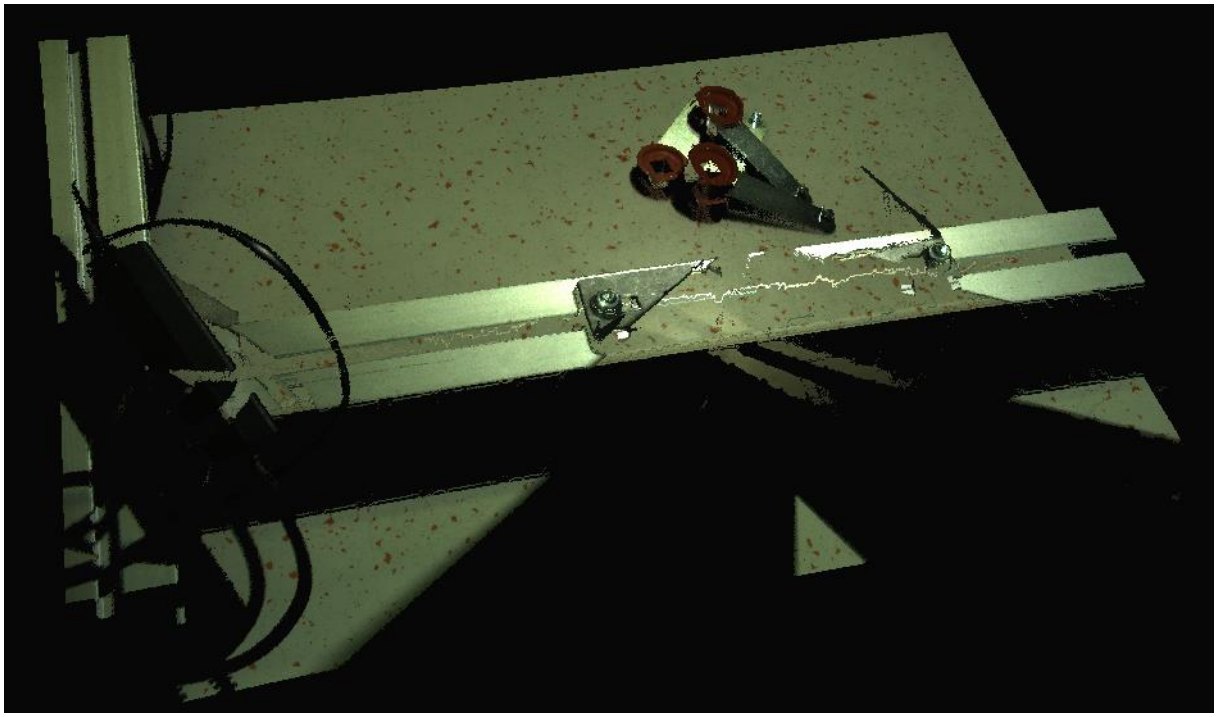
Slika 16. Serija slika kalibracijske ploče

Dobivena matrica transformacija H_{CAM}^{ROB} koristiti će se u poglavlju 6 gdje je opisana integracija cijelog sustava s robotom i algoritmom za lokalizaciju.

3.6. PCL(Point Cloud Library) Biblioteka

PCL je C++ biblioteka za obradu oblaka točaka. Biblioteka koristi FLANN(Fast Library for Approximate Nearest Neighbours) za svoje k-metode pretraživanja oblaka točaka. Biblioteka također koristi Boost pokazivače čime eliminira potrebu za višestrukim učitavanjem podataka već prisutnih u sustavu, za matematičke operacije koristi se Eigen biblioteka. Također je bitno naglasiti da PCL podržava paralelnu obradu podataka u procesoru, podržava SSE(Streaming SIMD Extensions) optimizaciju, dakle iskorištava svojstvo višenitnosti(*Multithreading*) modernih procesora. PCL ima mogućnosti: filtriranja, segmentacije, registracije i sl. [9].

Zivid kamera daje samo sirovi oblak točaka u prostoru(vidi sliku 17 za primjer). Taj oblak točaka sam po sebi nije previše koristan, potrebno je implementirati algoritme koji bi izvršavali prepoznavanje. Nad oblakom točaka koriste se razne operacije koje će biti opisane u ovom radu. Obrada oblaka točaka u ovom radu načelno je podijeljena na filtriranje i prepoznavanje, oblak se filtrira na različite načine kako bi se dobio što jednostavniji konačni oblak, koji bi u konačnici sadržavao samo dijelove scene od interesa za aplikaciju.



Slika 17. Oblak točaka dobiven Zivid One+ Medium kamerom

3.7. Prikaz značajki oblaka točaka

Kako bi se dobilo kvalitetno prepoznavanje objekata u oblaku točaka potrebno je prepoznati ključne točke u oblaku. Oblak točaka se ne može trivijalno promatrati kao skup točaka u prostoru, kada bi se oblak razmatrao samo kao skup točaka on bi bio gotovo neprimjenjiv. Točka sama po sebi ne opisuje kontekst njenog okruženja. Zato se kao osnovni gradivni element uzima tzv. lokalni deskriptor(*local descriptor*). Lokalni deskriptor može imati različite izvedbe kao npr. *shape descriptor*, *geometric feature* i sl. [10].

Formulacija lokalnog deskriptora za željenu točku interesa p_q može se definirati na sljedeći način, gdje je $P_k = \{p_1^k \,..\, p_2^k\}$ skup točaka u susjedstvu p_q. Susjedstvo se može definirati kao:

$$\|p_i^k - p_q\| \quad \leqq d_m \tag{11}$$

Gdje je d_m maksimalna dopuštena udaljenost susjedne točke od točke interesa. Također je moguće ograničiti broj susjeda k.

Tada se može definirati vektor F koji opisuje geometrijske informacije o skupu P_k oko točke p_q. Formulacija vektora deskriptora može glasiti:

$$F(p_q, P_k) = \{x_1 \;\; x_2 \;\; ..x_n \;\}\tag{12}$$

Tada umjesto uspoređivanja točaka mogu se uspoređivati vektori deskriptora.

Kao primjer može se uzeti Γ kao mjera sličnosti između dvije točke p_1 i p_2, gdje je d udaljenost između njih. Formulacija tada glasi:

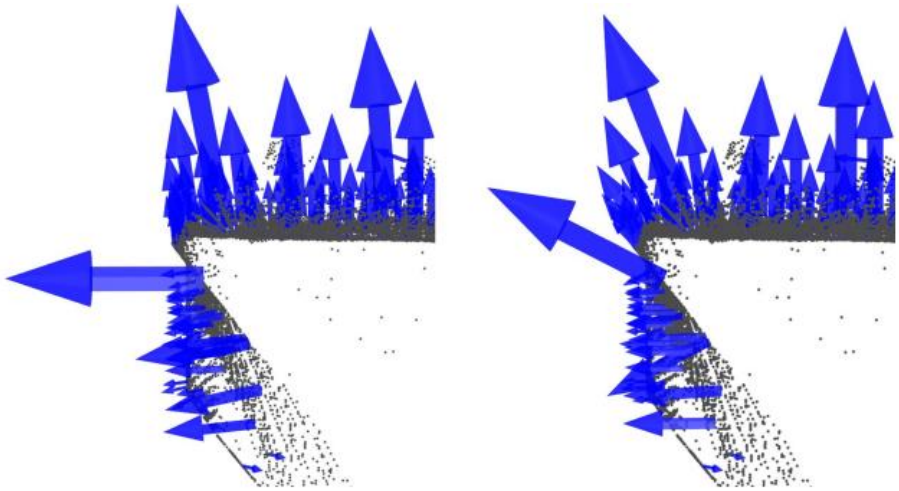
$$\Gamma = d(F_1, F_2) \tag{13}$$

Kada bi d između deskriptora dvije točke težio 0, te točke bi se mogle smatrati jednakima, ako je d prevelik tada te dvije točke nisu jednake. Na taj način uključujući susjedne točke opisuje se susjedna geometrija u formulaciji deskriptora. Željene karakteristike deskriptora su [10]:

- Otpornost na rigidne transformacije: 3D rotacije i translacije ne bi smjele utjecati na konačni vektor deskriptora F
- Neovisnost vektora F o gustoći uzorkovanja
- Otpornost na buku
- Vektor deskriptora mora održati jednaku ili sličnu formu u prisutnosti manjih smetnji u oblaku.

3.8. Estimacija normala

Da bi se odredili deskriptori potrebno je na neki način opisati geometriju površine oko točke od interesa. Da bi se odredila orijentacija i oblik površine koristi se estimacija normala. Metoda estimacije korištena u PCL biblioteci radi na principu aproksimacije normala okomitih na tangentu površine. Estimacija površine je problem metode najmanjih kvadrata, rješenje tj. potrebne normale dobivaju se analizom svojstvenih vektora i svojstvenih vrijednosti(PCA-*Principal Component Analysis*) matrice kovarijanci dobivene od susjednih točaka oko točke interesa. Broj susjednih točaka je ključan parametar za estimaciju normala. Ako se odabere previše točaka može doći do gubljenja informacija o svojstvima okolne geometrije oko točke interesa(npr. gubljenje rubova), a ako postoji premalo točaka isto tako može doći do krive estimacije normale i gubljenja sitnih detalja oblaka točaka. Primjer toga vidljiv je na slici 18. Lijevi primjer prikazuje dobro odabran broja susjednih točaka gdje se razumno dobro uočavaju detalji i dobro su naznačene rubovi, dok se na desnom primjeru koristi prevelik broj točaka, što dovodi do loše detekcije rubova [10].



Slika 18. Primjeri estimacije normala [10]

U ovom radu normale matrice se procjenjuju pomoću sljedećeg dijela koda na slici 19:

```
pcl::NormalEstimationOMP<PointType, NormalType> norm_est;
norm_est.setKSearch(15);

norm_est.setInputCloud(model);
norm_est.compute(*model_normals);

norm_est.setInputCloud(scene);
norm_est.compute(*scene_normals);
cout << "normals computed" << endl;
```

Slika 19. Kod za estimaciju normala

Gdje naredba *norm_est.setKSearch* uzima 15 najbližih točaka oko točke interesa kao referentni uzorak. Također postoje verzije algoritma gdje se umjesto broja točaka uzima radijus oko točke interesa, npr.1 mm.

3.9. Filteri

Filteri se koriste u svrhu olakšavanja daljnje obrade i dobivanje preciznijih vrijednosti prilikom kreiranja normala i deskriptora. U ovom radu pomoću filtara se smanjuje rezolucija oblaka, miču se *Outlier-i* i eliminiraju se dijelovi oblaka koji nisu od interesa za aplikaciju.

3.9.1. PassThrough

PassThrough filter služi za izdvajanje potrebnog djela scene. Npr. kamera će u slučaju ovog rada slikati ploču na kojoj se nalaze predmeti. Potrebno je iz dobivenog oblaka točaka maknuti sve dijelove koji nisu bitni za prepoznavanje, dakle u ovom slučaju to je sve ono što nije ploča na kojoj se nalaze predmeti, budući da na tim mjestima nema ništa od interesa za aplikaciju. *PassThrough* filter se implementira tako da se odrede granice i dimenzija na koju će se primijeniti(x,y ili z). U ovom radu primijenjen su tri iteracije filtra za svaku od dimenzija(x, y i z), primjer koda prikazan je na slici 20.

```
pcl::PassThrough<pcl::PointXYZRGB> pass;
pass.setInputCloud(cloud);
pass.setFilterFieldName("x");
pass.setFilterLimits(x0, x1);
// pass.setFilterLimitsNegative (true);
pass.filter(*cloud_filtered);

*cloud = *cloud_filtered;
pcl::PassThrough<pcl::PointXYZRGB> pass2;
pass2.setInputCloud(cloud);
pass2.setFilterFieldName("y");
pass2.setFilterLimits(y0, y1);
// pass.setFilterLimitsNegative (true);
pass2.filter(*cloud_filtered);

*cloud = *cloud_filtered;

*cloud = *cloud_filtered;
pcl::PassThrough<pcl::PointXYZRGB> pass3;
pass3.setInputCloud(cloud);
pass3.setFilterFieldName("z");
pass3.setFilterLimits(z0, z1);
// pass.setFilterLimitsNegative (true);
pass3.filter(*cloud_filtered);

*cloud = *cloud_filtered;
```

Slika 20. *Passthrough* filtar

Prednost korištenja *PassThrough* filtra je uklanjanje nepotrebnih informacija čime se olakšava obrada i smanjuje vrijeme procesa, nadalje smanjuje se mogućnost 'lažnih' korespondencija npr. pojave da algoritam pronalazi korespondencije na dijelovima scene gdje je nemoguće postojanje objekta od interesa.

3.9.2. Statistical Outlier Removal filtar

Prilikom dobivanja oblaka točaka i naknadne filtracije može doći do pojave udaljenih *outlier-a* koji mogu biti problem npr. u kontekstu dobivanja normala površine. Što u konačnici dovodi do problema s dobivanjem smislenih lokalnih deskriptora koji uzrokuju probleme s registracijom a samim time i prepoznavanjem. Jedan način da se prevenira ovakav slučaj je korištenje *Statistical Outlier Removal* filtra(na kojeg će se nekad referirat kao 'SOR' filtar u radu). SOR filtar se bazira na procjeni srednje aritmetičke vrijednosti udaljenosti točke od njenih susjednih točaka. Za svaku točku $p_q \in P^k$ gdje je P^k skup susjednih k točaka od točke interesa treba pronaći srednju udaljenost d_i između točke p_q i njezinih susjeda, treba izračunati prosječnu srednju udaljenost μ_d za cijeli oblak točaka, zatim izračunati standardnu devijaciju σ_d za srednju udaljenost d_i. Ako točka p_q ima vrijednost van granica određenih pomoću standardne devijacije σ_d i prosječne udaljenosti μ_d tada se ona uzima kao *outlier* i eliminira se iz izlaznog oblaka točaka [10]. Implementacija koda prikazana je na slici 21.

```
int SOR_Filter(pcl::PointCloud<pcl::PointXYZ>::Ptr &cloud, int MeanK)
{
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_filtered(new pcl::PointCloud<pcl::PointXYZ>);
    std::cerr << "Cloud before filtering: " << std::endl;
    std::cerr << *cloud << std::endl;
    pcl::StatisticalOutlierRemoval<pcl::PointXYZ> sor;
    sor.setInputCloud(cloud);
    sor.setMeanK(15);
    sor.setStddevMulThresh(1.0);
    sor.filter(*cloud);

    return (0);
}
```

Slika 21. Implementacija PCL *Statistical Outlier Removal* Filtra

U ovom radu je pomoću *sor.setMeanK(15)* naredbe broj susjednih točaka koji se uzimaju u obzir ograničen na 15, dok *sor.SetStddevMulThresh(1)* naredba uzrokuje eliminaciju svih točaka koje imaju udaljenost d_i veću od jedne standardne devijacije.

3.9.3. Downsampling

Smanjivanje broja točaka je ključno za brzinu obrade podataka,a također može zagladiti šum. U ovom radu korišten je *pcl::VoxelGrid* filter za koji se određuje dimenzija voksela(*leaf size*). Nakon određivanje dimenzije voksela unosi se ulazni oblak točaka koji se provlači kroz filter. Filter će za svaki voksel procijeniti jednu točku dobivenu kao srednju vrijednost susjednih točaka u vokselu. Implementacija koda prikazana je na slici 22.

```
int DownSampler(pcl::PointCloud<pcl::PointXYZ>::Ptr &cloud, float leafSize)
{
    pcl::VoxelGrid<pcl::PointXYZ> sor;
    sor.setInputCloud(cloud);
    sor.setLeafSize(leafSize, leafSize, leafSize);
    sor.filter(*cloud);
    return (1);
}
```

Slika 22. Implementacija *VoxelGrid* filtra

3.10. Segmentacija ravnine(*Plane Segmentation*)

Problem segmentacije ravnine rješava se *Random Sample Consensus metodom*(RANSAC) . Metoda se očituje u tome da se izaberu nasumično tri točke od ukupnog ulaznog oblaka točaka, od te tri točke estimira se ravnina. Zatim, nakon estimacije ravnine računaju se udaljenosti d svih točaka od navedene ravnine. Prebroji se broj točaka koje se nalaze u rasponu od $0 \leq |d| \leq |d_t|$, gdje je d_t maksimalna dozvoljena udaljenost *inlier-a* zadana od strane korisnika. Nakon što se pronađe najbolji osnovni model ravnine(onaj koji ima najviše *inlier-a*) računa se konačni model pomoću metode najmanjih kvadrata uzimajući u obzir sve *inlier-e* [10].

Nakon što se dobije konačni model ravnine pomoću *pcl::SACSegmentation* objekta izdvajaju se zagrade(*indices*) podataka koji sačinjavaju ravninu. pomoću *pcl::ExtractIndices* objekta izdvajaju se svi objekti osim ravnine, budući da nas sama ravnina ne zanima u kontekstu izuzimanja predmeta iz nestrukturirane okoline. Implementacija algoritma za izdvajanje površine vidljiva je na slici 23.

```
int PlaneSegmentation(pcl::PointCloud<pcl::PointXYZ>::Ptr &cloud)
{
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloudplane(new pcl::PointCloud<pcl::PointXYZ>);


    pcl::ModelCoefficients::Ptr coefficients(new pcl::ModelCoefficients);
    pcl::PointIndices::Ptr inliers(new pcl::PointIndices);

    pcl::SACSegmentation<pcl::PointXYZ> seg;
    seg.setOptimizeCoefficients(false);
    seg.setModelType(pcl::SACMODEL_PLANE);
    seg.setMethodType(pcl::SAC_RANSAC);
    seg.setDistanceThreshold(0.05);

    seg.setInputCloud(cloud);
    seg.segment(*inliers, *coefficients);

    if(inliers->indices.size() == 0)
    {
        PCL_ERROR("Nijedna ravnina nije pronadena");
        return (-1);
    }

    pcl::ExtractIndices<pcl::PointXYZ> extract;
    extract.setInputCloud(cloud);
    extract.setIndices(inliers);
    extract.setNegative(true);


    extract.filter(*cloud);
}
```

Slika 23. Implementacija RANSAC algoritma za segmentaciju ravnine

3.11. Euklidska klasterizacija(*Euclidian Clusterization*)

Zadatak euklidske klasterizacije je uzeti oblak točaka P i podjeliti ga u manje dijelove kako bi se vrijeme obrade oblaka P smanjilo.

U matematičkom smislu klaster je skup točaka koji zadovoljava sljedeći izraz:

$$O_i = \{p_i \in P\} \tag{14}$$

Koji se ne nalazi u skupu točaka:

$$O_j = \{p_j \in P\} \tag{15}$$

Za uvjet :

$$min\|p_i - p_j\|_2 \geq d_{th} \tag{16}$$

gdje je d_{th} maksimalan dozvoljen prag udaljenosti. Udaljenost d između dvije točke definira se kao euklidska udaljenost. Ako su udaljenosti d svih točaka između dva seta točaka $p_i \in P$ i $p_j \in P$ veće od d_{th} tada skup $p_i \in P$ pripada klasteru O_i a $p_j \in P$ pripada klasteru O_j, dakle potrebno je koristiti algoritam koji estimira minimalnu udaljenost između dva seta točaka. S ovom idejom, postoji algoritam u PCL biblioteci koji određuje klastere koristeći vrijednost d_{th} i minimalan broj točaka po klasteru, što može biti korisno ako je željena veličina klastera

unaprijed poznata npr. veličina klastera na sceni može se procijeniti prema veličini oblaka točaka modela [10]. Jedna od implementacija takvog algoritma na realnu aplikaciju prikazana je na slici 24.

```
void EuclidianClusterization(pcl::PointCloud<pcl::PointXYZ>::Ptr cloud)
{
    pcl::search::KdTree<pcl::PointXYZ>::Ptr tree(new pcl::search::KdTree<pcl::PointXYZ>);
    pcl::PointCloud<pcl::PointXYZ>::Ptr filtered_cloud(new pcl::PointCloud<pcl::PointXYZ>);
    std::vector<pcl::PointIndices> cluster_indices;
    tree->setInputCloud(cloud);
    pcl::EuclideanClusterExtraction<pcl::PointXYZ> ec;

    ec.setClusterTolerance(1); // 2cm
    ec.setMinClusterSize(10000);
    ec.setMaxClusterSize(35000); // 150000

    ec.setSearchMethod(tree);
    ec.setInputCloud(cloud);
    ec.extract(cluster_indices);
    for(std::vector<pcl::PointIndices>::const_iterator it = cluster_indices.begin(); it != cluster_indices.end(); ++it)
    {
        pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_cluster(new pcl::PointCloud<pcl::PointXYZ>);
        for(std::vector<int>::const_iterator pit = it->indices.begin(); pit != it->indices.end(); ++pit)
            cloud_cluster->push_back((*cloud)[*pit]); //*
        cloud_cluster->width = cloud_cluster->size();
        cloud_cluster->height = 1;
        cloud_cluster->is_dense = true;
        // if(cloud_cluster->size()>1000);
        *filtered_cloud += *cloud_cluster;
        std::cout << cloud_cluster->size() << std::endl;
    }
    pcl::copyPointCloud(*filtered_cloud, *cloud);
}
```

Slika 24. Kod za Euklidsku klasterizaciju

3.12. Prepoznavanje(*Recognition*)

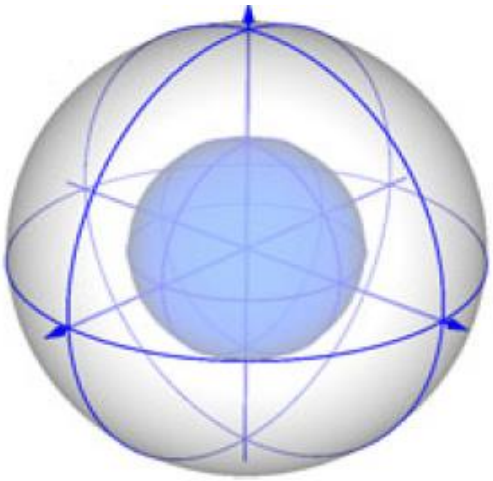
Program za prepoznavanje je vođen primjerom koda iz literature [11]. Postupak se temelji na tzv. *Correspondence Grouping* algoritmima koji za zadatak imaju klasteriranje *point to point* korespondencija između scene i modela nakon dobivanja lokalnih deskriptora. Nakon obavljenog klasteriranja *Correspondence Grouping* algoritmi daju 3D matricu transformacija za svaki od klastera tj. mogućih modela pronađenih na sceni.

Prije korištenja postupka prepoznavanja, na oblak točaka scene i modela implementiraju se prethodno navedeni filtri i segmentacija podloge. Također primjenjuje se estimacija normala koje će se primijeniti za određivanje deskriptora te za postupak klasteriranja.

3.12.1. Izračun SHOT deskriptora

SHOT(*Signature of Histograms of OrienTations*) je 3D deskriptor koji je robustan na smetnje, translacije i rotacije. Koristi RF lokalni referentni koordinatni sustav kako bi se osigurala robusnost. Struktura deskriptora sastoji se od izotropnog sfernog rastera koji je podijeljen u radijalnom smjeru, smjeru visine i u smjeru azimuta. Cijela sfera je podijeljena na 32 segmenta od kojih svaki sadrži lokalni histogram. Elementi lokalnih histograma računaju se na temelju

diferencijalnih značajki tj. u ovom slučaju normala na površinu. Skup izračunatih histograma tvori konačni deskriptor koji se koristi za opis geometrije objekta [12].



Slika 25. Prikaz SHOT deskriptora [12]

SHOT deskriptori se u ovom radu računaju pomoću korištenja ključnih točaka i normala. Ključne točke dobivaju se pomoću *pcl::UniformSampling<PointType>* objekta. Prikaz deskriptora nalazi se na slici 25, dok je kod za izračun deskriptora prikazan na slici 26.

```
//  Izračun deskriptora
//
pcl::SHOTEstimationOMP<PointType, NormalType, DescriptorType> descr_est;
descr_est.setRadiusSearch(descr_rad_);

descr_est.setInputCloud(model_keypoints);
descr_est.setInputNormals(model_normals);
descr_est.setSearchSurface(model);
descr_est.compute(*model_descriptors);

descr_est.setInputCloud(scene_keypoints);
descr_est.setInputNormals(scene_normals);
descr_est.setSearchSurface(scene);
descr_est.compute(*scene_descriptors);
cout << "descriptors computed" << endl;
//
```

Slika 26. Izračun SHOT deskriptora

3.12.2. Pronalaženje korespondencija između modela i scene

Kada postoje scena i model s pripadajućim deskriptorima potrebno je na neki način pronaći relacije između ta dva oblaka. To se postiže korištenjem k-d stabla.

K-d stablo je u PCL-u implementirano pomoću FLANN biblioteke i glasi *pcl:KdTreeFLANN<DescriptorType>*. U objekt tj. instancu ove klase unosi se ulazni oblak točaka koji sačinjavaju deskriptori modela. Svrha toga je da k-d stablo pronađe najbliži deskriptor na ulaznom oblaku modela za svaki deskriptor na sceni. To se postiže iterativnim postupkom traženja kroz svaku ključnu točku scene, za svaku točku scene traži se najprikladniji

deskriptor u oblaku modela, ako taj najbliži deskriptor u modelu ne prelazi potrebnu kvadratnu granicu udaljenosti npr. 0.25(udaljenosti u SHOT deskriptorima su uglavnom između 0 i 1, gdje je 0 savršeno poklapanje) tada se ta korespondencija sprema u objekt klase *pcl::CorrespondencePtr* [10],[11]. Primjer koda implementiranog u C++ za pronalazak korespondencija vidljiv je na slici 27.

```
pcl::CorrespondencesPtr model_scene_corrs(new pcl::Correspondences());

pcl::KdTreeFLANN<DescriptorType> match_search;
match_search.setInputCloud(model_descriptors);

for(std::size_t i = 0; i < scene_descriptors->size(); ++i)
{
    std::vector<int> neigh_indices(1);
    std::vector<float> neigh_sqr_dists(1);
    if(!std::isfinite(scene_descriptors->at(i).descriptor[0]))
    {
        continue;
    }
    int found_neighs = match_search.nearestKSearch(scene_descriptors->at(i), 1, neigh_indices, neigh_sqr_dists);
    if(found_neighs == 1
       && neigh_sqr_dists[0]
              < 0.25f)
    {
        pcl::Correspondence corr(neigh_indices[0], static_cast<int>(i), neigh_sqr_dists[0]);
        model_scene_corrs->push_back(corr);
    }
}
```

Slika 27. Pronalaženje korespodencija između modela i scene

3.12.3. Klasteriranje korespodencija i lokalizacija

Nakon pronalaska korespondencija koje su spremljene pomoću objekta tipa *pcl::CorrespondencePtr* potrebno je klasterirati točke oblaka pomoću pronađenih korespondencija. U ovom radu korišten je Hough-ov algoritam glasovanja pomoću klase *pcl::Hough3DGrouping<PointType, PointType, RFType, RFType>*. Također je potrebno naglasiti da svaka ključna točka mora imati lokalni koordinatni sustav za korištenje Hough-ovog algoritma. Kao ulaz u algoritam koriste se lokalni koordinatni sustavi ključnih točaka i njihovi deskriptori za model i scenu, također je potreban prethodno dobiven vektor korespondencija. Kao izlaz algoritam daje pronađene klastere gdje uz svaki dolazi pripadajuća matrica homogenih transformacija pronađene instance modela na sceni u odnosu na ulazni model koji se uspoređuje sa scenom, dobivanjem matrice transformacija završena je početna lokalizacija.

3.12.4. Povećanje točnosti lokalizacije pomoću ICP algoritma

Budući da Hough-ov algoritam glasanja daje grube matrice transformacija, gdje točnost obično ima varijacije od nekoliko milimetara do nekoliko desetaka milimetara u translaciji, isto tako postoji mogućnost varijacije matrice rotacija. Da bi se dobili bolji rezultati, primjenjivi na

realnoj aplikaciji koristi se *Iterative Closest Point(ICP)* algoritam koji nastoji smanjiti ukupnu euklidsku udaljenost između dva seta oblaka točaka kako bi u konačnici algoritam konvergirao i dao matricu transformacija između dva oblaka točaka. Kao ulaz koriste se oblaci točaka scene i modela(koji je pomaknut za matricu transformacija dobivenu u postupku iz podpoglavlja 3.12.3). Kao izlaz dobiva se matrica transformacija.

Ovaj algoritam za manu ima mogućnost da ostane u lokalnom optimumu, zato on sam nije praktičan za lokalizaciju. Da bi ICP bio koristan potrebno je prvo dobiti grubo pozicioniranje uz pomoć algoritma za klasteriranje iz 3.12.3, nakon čega bi ICP algoritam fino pozicionirao prethodno dobiveni oblak točaka. Primjer implementacije ICP algoritma vidljiv je na slici 28.

```
pcl::IterativeClosestPoint<PointType, PointType> icp;
icp.setMaximumIterations(icp_max_iter_);
icp.setMaxCorrespondenceDistance(icp_corr_distance_);
icp.setInputTarget(scene);

icp.setInputSource(instances[main_instance]);
pcl::PointCloud<PointType>::Ptr registered(new pcl::PointCloud<PointType>);
icp.align(*registered);
registered_instances.push_back(registered);
if(icp.hasConverged())
{
    std::cout << "*----------------*" << std::endl;
    std::cout << "Aligned!" << std::endl;
    ICPrototranslation = icp.final_transformation_;
}
else
{
    std::cout << "*----------------*" << std::endl;
    std::cout << "Not Aligned!" << std::endl;
}
```

Slika 28. ICP algoritam

4. Robotski sustav

Nakon lokalizacije predmeta dobiveni podatci o poziciji i orijentaciji predmeta se prenose na robota i njihova točnost se provjerava na fizičkom postavu. U ovom radu korišten je robotski manipulator CSDA10F proizvođača Yaskawa Electric Corporation(vidi sliku 29), robot je uparen sa kontrolerom FS100(vidi sliku 30).

Slika 29. Yaskawa CSDA10F 15-osni manipulator [13]

Slika 30. FS100 kontroler [13]

4.1. Pregled sustava

4.1.1. CSDA10F manipulator

CSDA10F je 15-osni manipulator osmišljen za laboratorijsku primjenu. Svaka ruka ima 7 osi uz jednu središnju okretnu os koja robotu omogućuje jednostavnu promjenu radne stanice i znatno povećava radni opseg robota. Prednost robota su tvornički provedene cijevi za pneumatiku i signalni kablovi, tako da je potreba za dodavanjem dodatnih nosača kablova na robota minimalna, što bi znatno smanjilo mobilnost robota. Također manipulator ima zabrtvljenu konstrukciju koja zadovoljava IP64 standard dok okretna os u bazi robota zadovoljava standard IP50, što ga čini primjenjivim u agresivnim okolinama. Treba napomenuti da je premazan posebnim zaštitnim slojem koji mu omogućuje da se manipulator čisti vodikovim peroksidom što dodatno povećava mogućnost osjetljive laboratorijske primjene [13].

4.1.2. FS100 kontroler

FS100 je kontroler osmišljen za fleksibilne aplikacije i manipulatore sa nosivostima do 20kg. Ima mogućnost upravljanja sa 4 do 15 osi ovisno o konfiguraciji(u ovom slučaju 15 osi). Posjeduje ARM(*Advanced Robot Motion Function*) funkciju koja osigurava visoku točnost programirane putanje, optimalne kretnje, brzinu robota i visoko osjetljivu detekciju kolizije. Također ima mogućnost korištenja MotoPlus paketa, što dodatno poboljšava mogućnost komunikacije i obrade podataka na samom kontroleru [14],[15].

4.1.3. INFORM

Primarni programski jezik koji upravlja robotom, vidljiv je na *Teach Pendantu* i može se mijenjati pomoću istog. Sadrži naredbe za kretnje robota, postavljanje izlaza, čitanje ulaza, promjenu varijabli robota i sl. Također jedna od svrha mu je upravljanje tokom programa korištenjem petlji npr. FOR. Jezik se koristi unutar definirane cjeline naziva *Job*. *Job* cjeline predstavljaju procedure koje koristi robot, one mogu biti u svrsi kretnje robota, komunikacije, organizacije toka programa i sl. *Job* cjeline se mogu modificirati pomoću tekstualnog editora. Također postoji mogućnost pozivanja *Job* cjelina unutar drugih *Job* cjelina što omogućuje lakšu i pregledniju organizaciju procedura.

4.1.4. Motoplus SDK

Motoplus SDK pruža mogućnost razvoja aplikacija u C programskom jeziku za FS100 kontroler, te aplikacije rade u pozadini, u realnom vremenu, paralelno s normalnim radom kontrolera. Omogućava fleksibilnu integraciju robota s okolinom budući da posjeduje

mogućnost *Ethernet socket* komunikacije s vanjskim uređajima. Također posjeduje razna sučelja prema kontroleru kao npr. Prijenos podataka, sučelje prema varijablama i registrima kontrolera, sučelje prema I/O komunikaciji, sučelje prema *Job* podacima i pokretanje istih, mogućnost definiranja putanja i pokretanje gibanja robota i sl. [16].

Aplikacije se mogu razvijati u MotoPlusIDE programskom okruženju ili u Visual Studio-u gdje se Motoplus koristi kao *plug-in*, u Visual Studio-u je Motoplus kompatibilan sa proširenjima IntelliSense®, Git, TFS i sl. što znatno olakšava programiranje [16].

Debuggiranje aplikacije vrši se pomoću Telnet aplikacije na računalu, gdje su kontroler i računalo povezani *Ethernet* kabelom i nalaze se na istoj mreži(*Subnet*).

5. Izrada eksperimentalnog postava

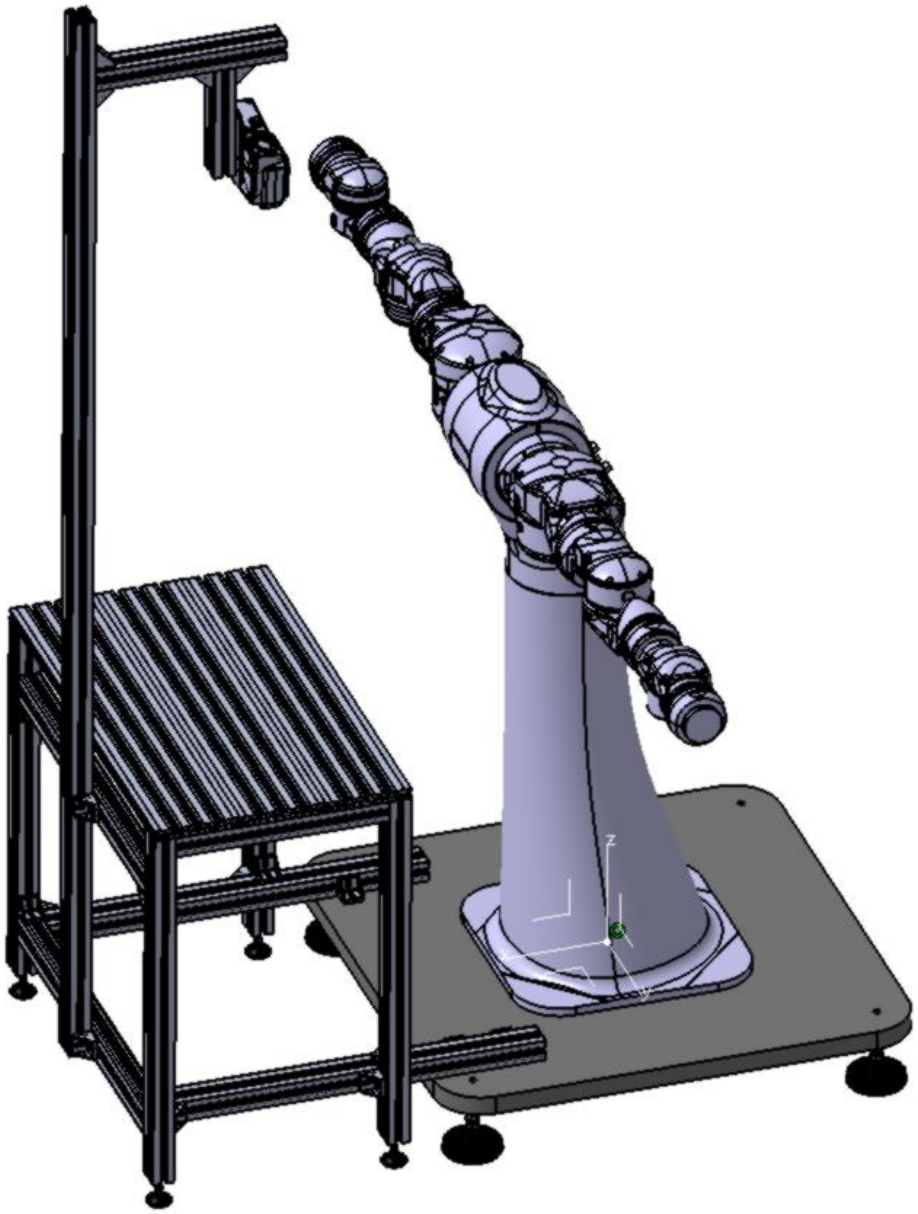
5.1. Schunk SWS 011

Schunk SWS 011 je sustav za izmjenu alata na robotskoj prihvatnici. Princip je da se jedan dio sklopa(*Master*) pričvrsti na prirubnicu robota, dok bi se drugi dio sklopa *slave* pričvrstio na korišteni alat. Na taj način dobiva se sustav na kojemu je moguće izvesti automatsku izmjenu alata. Prednosti ovog sustava su mogućnost ručnog odvajanja alata, visoka krutost(funkcionalne komponente izrađene su od tvrdog čelika), mogućnost dodavanja komunikacijskog modula, žljebovi za pneumatiku(uključujući i vakuum) i mogućnost provođenja tekućina kroz sustav izmjenjivača(ako bi aplikacija zahtijevala to). Također treba naglasiti da sustav ima ugrađen sigurnosni mehanizam u smislu da radi na NC principu [17]. Sklop SWS 011 prikazan je na slici 31. Ovaj izmjenjivač alata služi za pričvršćivanje ploče za kalibraciju i vakuumske prihvatnice na robota.

Slika 31. Schunk SWS 011 [17]

5.2. Prijedlog radne stanice

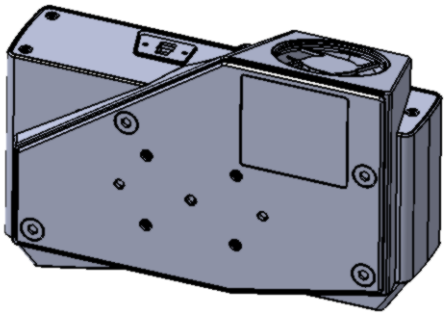
Specifično za ovog robota je što ima petnaestu os na svojoj bazi, dakle može se potpuno zakrenuti oko nje. Zbog te prednosti ideja je da se naprave identične stanice sa više strana robota. Jedna od tih stanica je izrađena i korištena u ovom diplomskom radu, može se vidjeti na slici 32.



Slika 32. Eksperimentalni postav

Stanica je konstruirana pomoću Bosch Rexroth profila te je pomoću pravokutnih kutnika pričvršćena za temeljnu ploču, s druge strane s plastičnim stopama se oslanja na pod. Na stol robota pričvršćen je nosač za Zivid 3D kameru, korištenje profila osigurava dostatnu krutost cijele konstrukcije koja drži kameru, budući da je to od ključne važnosti. Točna pozicija kamere u prostoru nije od ključne važnosti, taj problem se rješava pomoću *Hand Eye* kalibracije. Bitnije je osigurati krutost pozicije kamere u prostoru, kamera mora ostati statična u slučaju udara robota ili drugih objekata u konstrukciju kako bi lokalizacija predmeta bila uspješna. Cijeli postav je temeljen na modularnom spajanju profila pomoću kutnika, time se postiže mogućnost promjene pozicije kamere u sve tri osi (x, y, z) budući da različite pozicije kamere mogu imati različite kvaliteta prikupljenih oblaka točaka. Također dijelovi različitih dimenzija mogu zahtijevati različitu visinu kamere u odnosu na radni prostor robota, veći dijelovi zahtijevaju manju rezoluciju ali i veće vidno polje pa je kameru potrebno postaviti na višu poziciju. Dakle pozicija kamere ovisi o aplikaciji i komadu koji se želi prepoznati u oblaku točaka. Raspored rupa za montažu kamere može se vidjeti prema CAD modelu kamere na slici 34. Kamera je na nosivu konstrukcija pričvršćena pomoću adapterske gdje čime je pozicionirana na postavu, postav u stvarnosti se može vidjeti na slici 33.

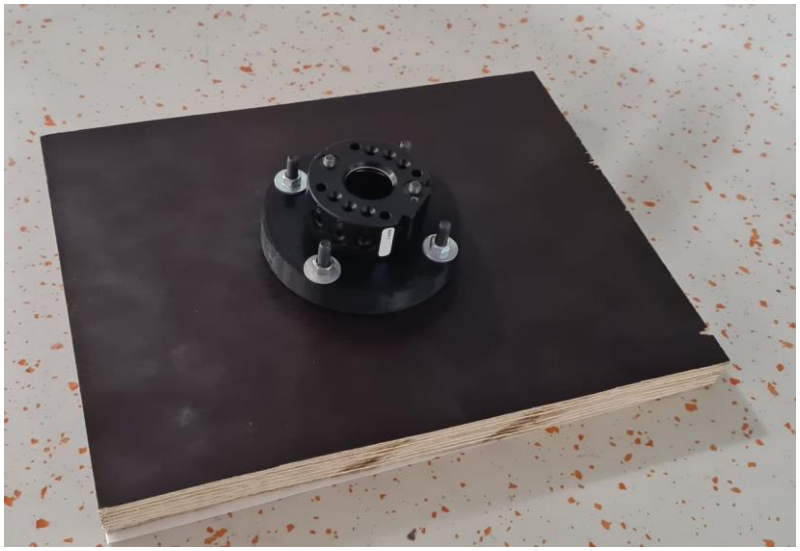
Slika 33. Stvarni postav



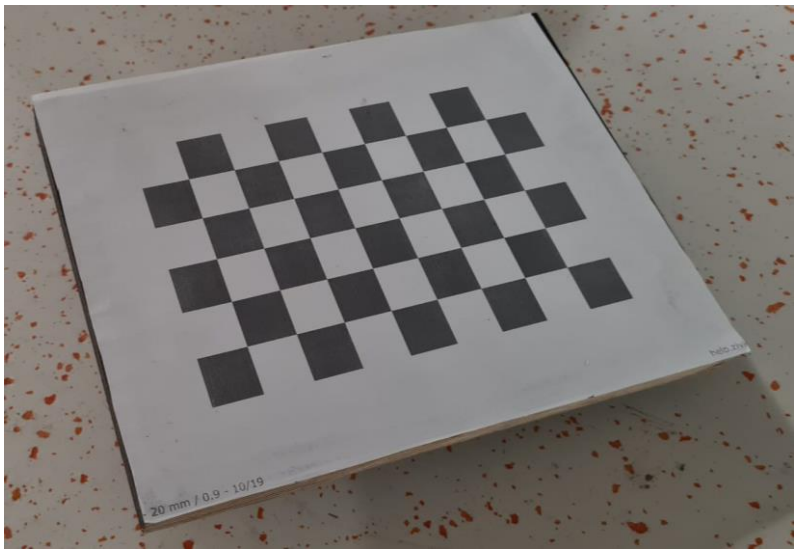
Slika 34. CAD Model kamere

5.3. Sklop kalibracijske ploče

Da bi se izvela kalibracija kamere potrebno je na neki način pričvrstiti kalibracijsku podlogu na sustav robotske ruke. U svrhu toga predložen je sklop za adaptersku ploču , prikaz se nalazi na slikama 35 i 36. Sklop je kompatibilan sa prethodno navedenim izmjenjivačem alata.



Slika 35. Sklop kalibracijske ploče 1



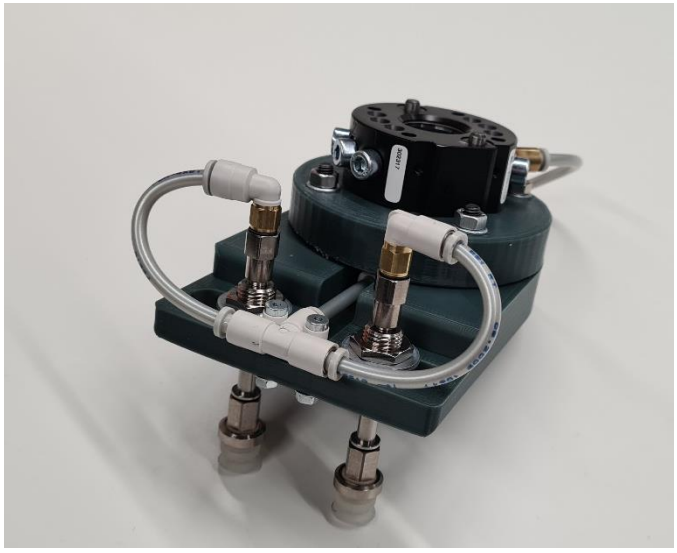
Slika 36. Sklop kalibracijske ploče 2

5.4. Vakuumska prihvatnica

Vakuumska prihvatnica je izrađena koristeći SMC-ove komponente u kombinaciji sa tehnologijom 3D printanja. Izrađena je iz dva dijela, gdje jedan služi kao prirubnica na *Slave* modul izmjenjivača alata iz podpoglavlja 5.1., dok drugi na sebi ima montiran vakuum ejektor ZU07SA(vidi sliku 37) i prihvatnice sa kompenzacijom hoda SMC ZPB2J10-B5(vidi sliku 38). Gotova prihvatnica vidljiva je na slici 39.

Slika 37. Vakuum ejektor ZU07SA

Slika 38. Prihvatnica s kompenzacijom hoda SMC ZPB2J10-B5



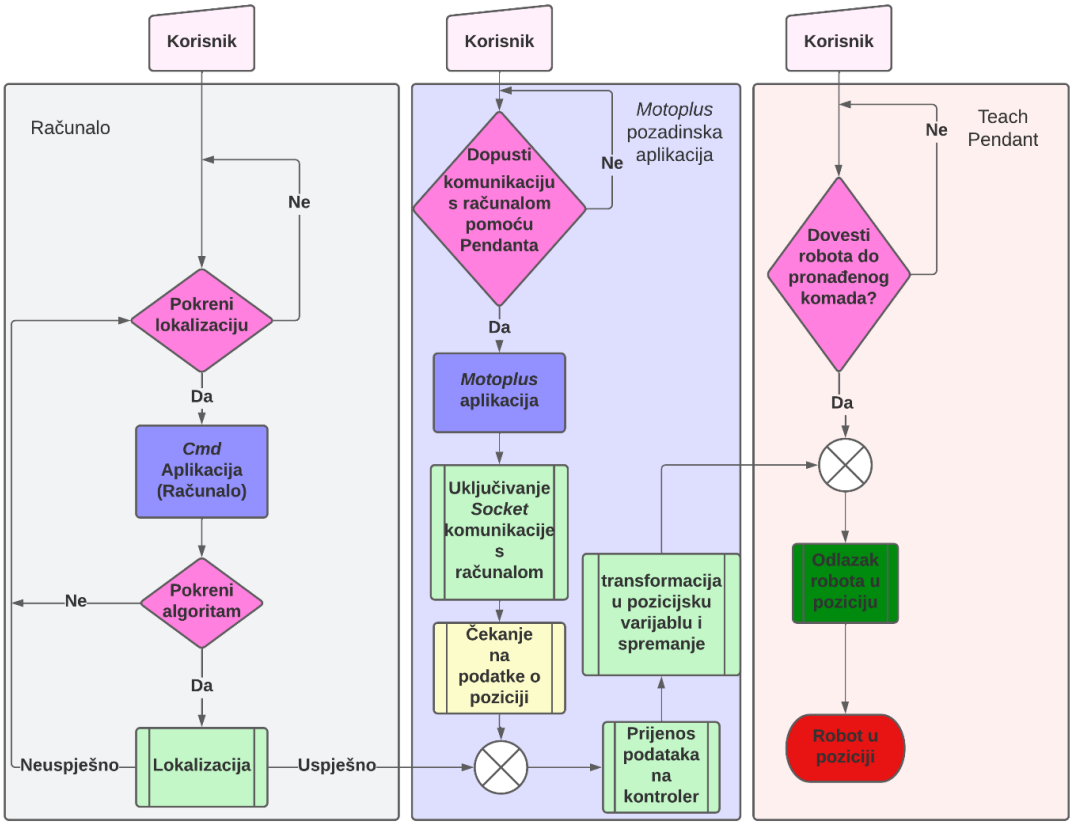
Slika 39. Vakuumska prihvatnica

6. Realizacija lokalizacijskog sustava

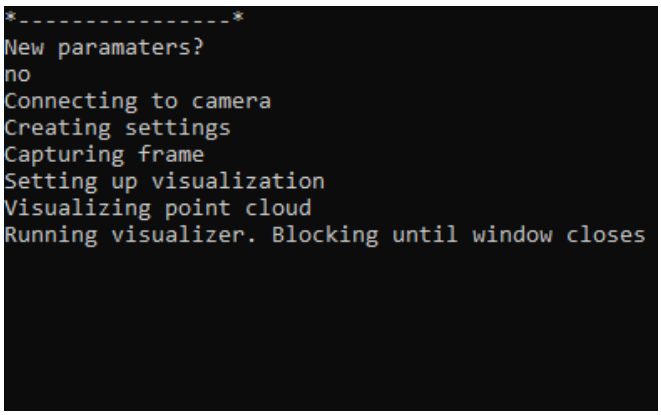
Cijeli sustav osmišljen je tako da se može jednostavno koristiti neovisno o objektu traženja. Ako bi se promijenio objekt traženja bilo bi potrebno neznatno modificirati postojeći sustav, gdje bi svaka modifikacija bila rutinska tj. bilo bi potrebno podesiti parametre u algoritmu traženja i ponovno umjeriti model na robotskom manipulatoru.

Sustav se sastoji od *Command prompt* aplikacije koja koristi Zivid kameru i algoritam za lokalizaciju koji se povezuje preko *Socket* komunikacije na pozadinsku Motoplus aplikaciju kontrolera FS100. Motoplus aplikacija od *Command prompt* zaprima informacije o pronađenom predmetu i pretvara tu informaciju u pozicijsku varijablu koju kontroler može iskoristiti za odlazak u poziciju u kojoj je predmet pronađen. Motoplus aplikacija se također koristi za početno umjeravanje objekta traženja, o tome će se više reći u nastavku. Nakon spremanja informacija u pozicijski registar od strane Motoplus aplikacije, korisnik može pomoću *Job-a* odvesti manipulator u željenu poziciju. Korisnik pomoću univerzalnih I/O signala na *Teach Pendantu* zadaje Motoplus aplikaciji naredbe koju će funkciju koristiti npr. čitanje poziciji, spremanje pozicije, komunikacija s računalom i sl. Dijagram toka sustava prikazan je na slici 40.

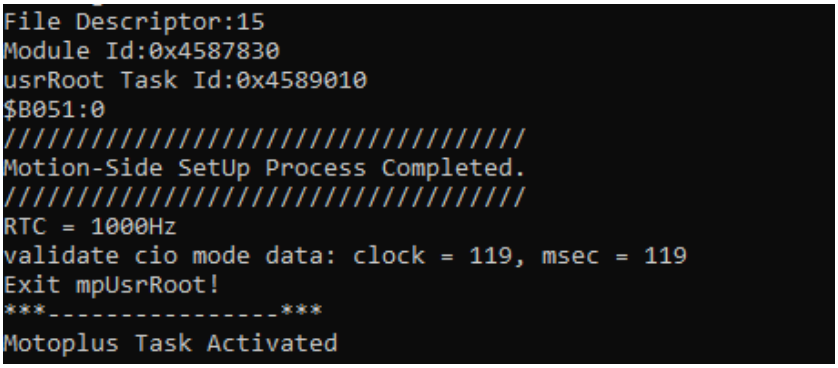
Također Motoplus aplikacija ima funkciju za slanje matrica homogenih transformacija potrebnih pozicijskih varijabli korištenih u *Hand Eye* kalibraciji pomoću Telnet aplikacije na računalu. Telnet aplikacija omogućava korisniku da vrši ispis i prikaz informacija na zaslonu računala od strane Motoplus aplikacije(pomoću funkcija *puts()* i *printf()* korištenih u Motoplus aplikaciji). Prikazi sučelja *Cmd* aplikacije i Telnet-a koji komunicira s Motoplus aplikacijom mogu se vidjeti na slikama 41 i 42.



Slika 40. Dijagram toka cjelokupnog sustava



Slika 41. *Cmd* aplikacija



Slika 42. Pokretanje Motoplus aplikacije

6.1. Sustav za lokalizaciju

Sustav za lokalizaciju sastoji se od Zivid kamere i algoritama za pronalaženje komada na sceni. Dijelovi algoritma za lokalizaciju opisani su u poglavlju 3 ali budući da je proces relativno složen, najlakše ga je prikazati redno, pomoću koraka, što je i napravljeno ispod.

Proces za lokalizaciju sastoji se od:

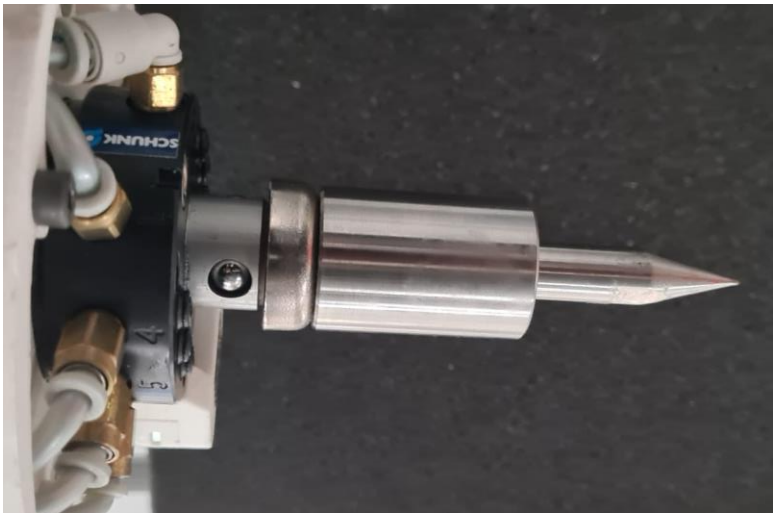
1. Uzimanja oblaka točaka(scene) pomoću Zivid kamere i učitavanje oblaka točaka modela
2. Pretvorba Zivid .zdf formata u .pcd format prihvatljiv za PCL biblioteku.
3. Korištenje *PassThrough* filtra kako bi se izolirao prostor od interesa na sceni
4. Primjena *Statistical Outlier Removal* Filtra kako bi se uklonio šum sa scene
5. Segmentacija ravnine i izbacivanje iste sa scene
6. Euklidska klasterizacija scene i dobivanje klastera koji će biti lakši za obradu, algoritam na sceni ostavlja samo pronađene klastere
7. Estimacija normala na sceni i modelu
8. *Downsampling* modela i scene te izlučivanje ključnih točaka
9. Izračun SHOT deskriptora scene i modela pomoću ključnih točaka i normala modela i scene
10. Pronalazak korespodencija između modela i scene pomoću Kd-stabla
11. Klasteriranje korespodencija pomoću Hough-ovog algoritma i dobivanje početnih matrica transformacija H_{Init}
12. Odabire se klaster s najviše korespodencija kao onaj koji će se izuzeti
13. Odabrani klaster provodi se kroz ICP algoritam gdje se dobiva matrica transformacija H_{ICP} koja preciznije određuje poziciju pronađenog objekta, kao početna pretpostavka uzima se pozicija pronađena pomoću Hough-ovog algoritma u koraku 11
14. Verifikacija točnosti lokalizacije pomoću *pcl::visualization::PCLVisualizer viewer,* u *viewer-u* se postave početni oblak scene, referentni model i poravnati model(model na koji su primjenjene transformacije H_{Init} i H_{ICP} . Ako se poravnati model poklapa sa onim na sceni može se pretpostaviti da je lokalizacija uspješna, ako nije, kreće se opet od koraka 1
15. Transformacija matrica H_{Init} i H_{ICP} u matricu H_{NEWOBJ}^{ROB} i slanje na kontroler manipulatora.

6.2. Uzimanje oblaka točaka i umjeravanje početne pozicije modela

U podpoglavlju 6.1 podrazumijeva se da postoji oblak točaka modela. Oblak točaka modela ne dobiva se *online* kao oblak scene, oblak modela prethodno se uzima zasebno, te obrađuje i umjerava u odnosu na manipulator na sljedeći način:

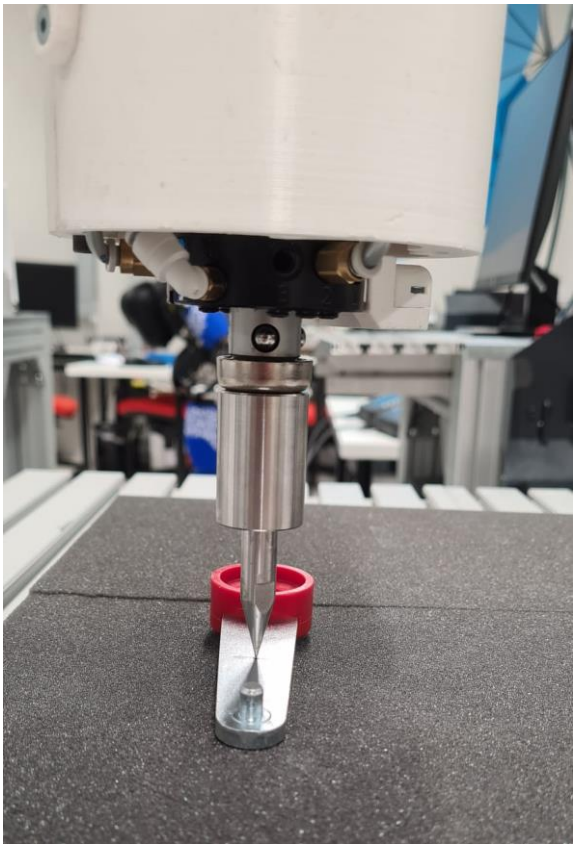
1. Uzimanje oblaka točaka pomoću Zivid kamere
2. Primjena *PassThrough* i *Statistical Outlier Removal* filtara. Segmentacija i otklanjanje ravnine ispod modela
3. Ako model nije potpuno izoliran od okoline, ponoviti prethodna dva koraka
4. Spremiti model u .pcd formatu

Nakon ekstrakcija oblaka točaka modela potrebno je odrediti referentnu poziciju robota u odnosu na model. To se radi na način da se TCP(*Tool Center Point*) alata dovede u željenu poziciju i orijentaciju u odnosu na početni model. U ovom slučaju TCP će biti točka na vrhu potokarenog stožca prikazanog na slici 43.



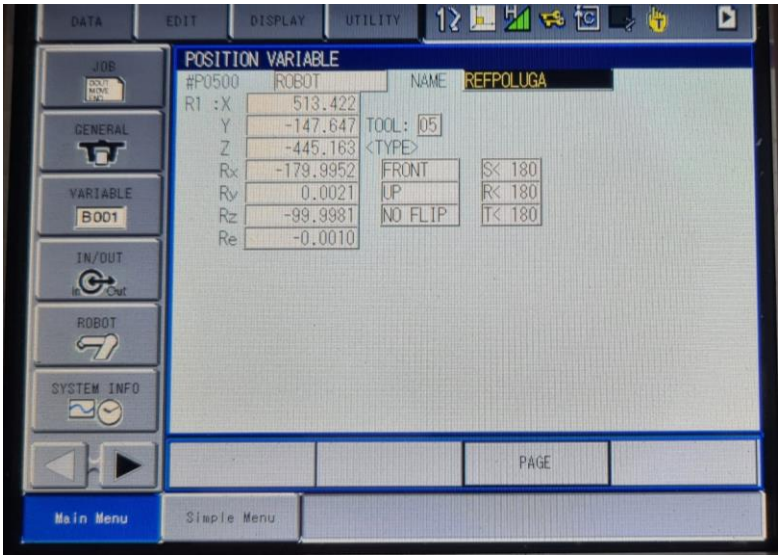
Slika 43. Korišteni TCP za umjeravanje

Pozicija i orijentacija ruke manipulatora u odnosu na referentni model prikazana je na slici 44.



Slika 44. Robot u referentnoj točci sa TCP-om alata

Nakon dovođenja u željenu poziciju, pozicija se sprema u pozicijski registar, u formatu gdje je pozicija opisana koordinatama x, y i z u baznom koordinatnom sustavu robota, a orijentacija Eulerovim kutovima oko vrha alata(R_x, R_y i R_z). U kontroleru FS100 pozicijska varijabla referentne pozicije modela prikazana je na slici 45 i nalazi se u registru 500.

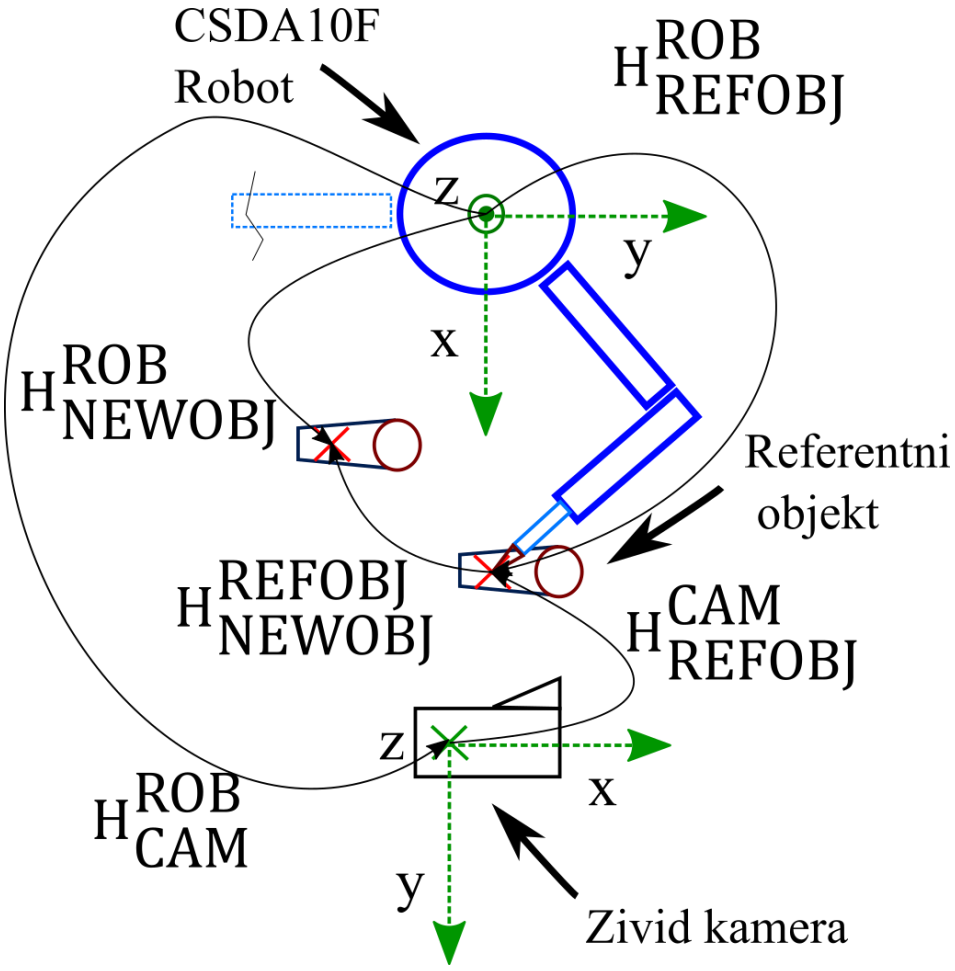


Slika 45. Prikaz referentne pozicije u pozicijskom registru

Nakon spremanja, referentna pozicija se pomoću funkcije izrađene pomoću Motoplus SDK šalje na Telnet sučelje na računalu spojenog na mreži s kontrolerom. Funkcija za slanje pozicije 500 na Telnet aktivira se aktivacijom univerzalnog I/O signala pomoću *Teach Pendanta* od strane korisnika, u ovom slučaju to je signal 10030. Na aktivaciju I/O signala funkcija radi transformaciju pozicijske varijable u homogenu matricu transformacija između baze robota i referentne točke modela koja se šalje na Telnet sučelje. Nakon toga korisnik ručno sprema dobivenu matricu transformacija i dodaje ju u *placeholdere* unutar algoritma za lokalizaciju. Dobivena matrica transformacija označavat će se kao H_{REFOBJ}^{ROB}.

6.3. Povezivanje sustava za lokalizaciju sa manipulatorom

Na slici 46 prikazana je shema postava sa svim relevantnim matricama transformacija i članovima sustava.



Slika 46. Shema postava

Da bi se odredili međusobni odnosi elemenata sustava potrebno je definirati svaku od matrica transformacija i način na koji se dobiva.

- H_{CAM}^{ROB} ili matrica transformacija od baznog koordinatnog sustava robota do ishodišta koordinatnog sustava kamere
  - Dobiva se *Hand Eye* kalibracijom opisanom u poglavlju 3.
- H_{REFOBJ}^{ROB} ili matrica transformacija od baznog koordinatnog sustava robota do referentnog objekta koji predstavlja model u algoritmu za lokalizaciju
  - Dobiva se dovođenjem TCP-a alata u referentnu točku modela i spremanjem te točke u pozicijsku varijablu, koja se transformira u matricu homogenih transformacija i šalje na računalo pomoću Motoplus aplikacije.

- H_{REFOBJ}^{CAM} ili matrica transformacija od ishodišta kamere do referentnog objekta
  - Dobiva se sljedećim izrazom:

$$H_{REFOBJ}^{CAM} = (H_{CAM}^{ROB})^{-1} \cdot H_{REFOBJ}^{ROB} \quad (17)$$

- H_{NEWOBJ}^{REFOBJ} ili matrica transformacija od modela(referentnog objekta) do pronađenog objekta na sceni pomoću algoritma lokalizacije
  - Dobiva se sljedećim izrazom(gdje su članovi opisani u podpoglavlju 6.1):

$$H_{NEWOBJ}^{REFOBJ} = H_{ICP} \cdot H_{Init} \quad (18)$$

- H_{NEWOBJ}^{ROB} ili matrica transformacija od baznog koordinatnog sustava robota to pronađenog objekta na sceni.
  - Dobiva se sljedećim izrazom:

$$H_{NEWOBJ}^{ROB} = H_{CAM}^{ROB} \cdot \left(H_{NEWOBJ}^{REFOBJ} \cdot H_{REFOBJ}^{CAM}\right) \quad (19)$$

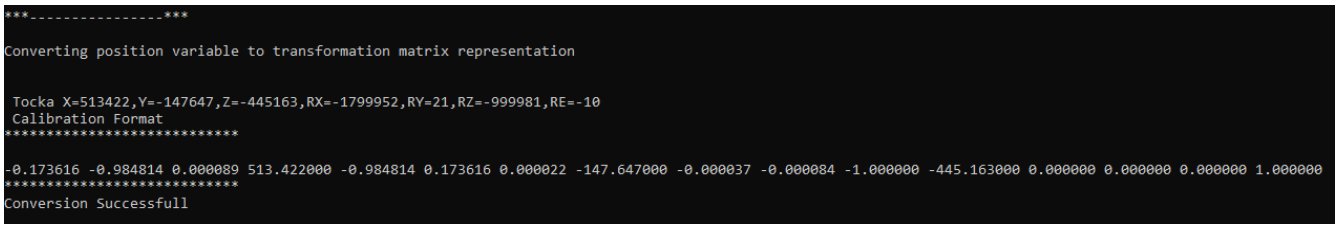
Razlog zašto se H_{NEWOBJ}^{REFOBJ} nalazi ispred člana H_{REFOBJ}^{CAM} na početku jednadžbe(19) je taj što algoritam za lokalizaciju daje apsolutne transformacije, dok se u ostatku rada koriste relativne transformacije.

6.4. Pregled funkcija Motoplus aplikacije

U poglavlju 6 opisan je način rada cjelokupnog sustava. Opisana je uloga Motoplus aplikacije i navedene su njezine funkcije. U ovom podpoglavlju dodatno će se objasniti tri glavne funkcije aplikacije i način na koji rade.

6.4.1. Slanje pozicije referentnog objekta na računalo

Kao što je već spomenuto, slanje referentne pozicije na računalo izvršava se ručnom aktivacijom signala 10030 na *Teach Pendantu*. Motoplus aplikacija pomoću funkcije *mpReadIO()* prepoznaje aktivaciju signala i pokreće funkciju. Pokretanjem funkcije iščitava se pozicijska varijabla 500 iz svog registra pomoću funkcije *mpGetUserVars()*. Uzimaju se vrijednosti pozicije x, y, z i orijentacija R_x, R_y, R_z i R_e, te se pomoću funkcije *mpZYXeulerToFrame()* transformiraju u matricu homogenih transformacija. Nakon transformacije članovi matrice transformacija šalju se prema Telnet sučelju na računalu. Prikaz transformirane varijable stanja na Telnet sučelju može se vidjeti na slici 47.



Slika 47. Prikaz transformirane varijable stanja na Telnet-u

Opisana funkcija je nazvana *SendPose()* i njezin kod je dan kao primjer, može se vidjeti na slici 48. Također treba naglasiti da je tokom programiranja u Motoplus biblioteci korisno koristiti *printf()* i *puts()* naredbe kao povratnu vezu od kontrolera, budući da sam Motoplus ne izbacuje greške za svoje integrirane funkcije(npr. *mpGetUserVars()*). On jedino šalje 0 ili -1 nakon izvršene funkcije ovisno o uspjehu, ako je funkcija uspješno izvršena Motoplus šalje 0, ako je neuspješno izvršena Motoplus šalje -1. Neovisno jeli povratna informacija 0 ili -1, program će se nastaviti izvršavati. Tako naprimjer ako funkcija Motoplusa *mpWriteIO()* ne uspije u svom zadatku poslat će -1 kao odgovor i program će nastaviti sa sljedećom naredbom, ako nije predviđen *Error Handling,* može doći do posljedica(npr. kolizije).

```
STATUS SendPose(UINT32 ioAddr)
{
    MP_IO_INFO dta;
    USHORT rSignalInfo;
    MP_USR_VAR_INFO PoseInfo;
    MP_COORD FromCoord;
    MP_FRAME ToFrame;
    LONG resp1;
    dta.ulAddr = ioAddr;
    resp1 = mpReadIO(&dta, &rSignalInfo, 1);

    if (rSignalInfo == 1)
    {
        puts("***----------------***\n");
        SetIO(dta.ulAddr, 0);
        mpTaskDelay(delayL);
        PoseInfo.var_type = MP_VAR_P;
        PoseInfo.var_no = 500;
        resp1 = mpGetUserVars(&PoseInfo);
        if (resp1 == 0)
        {
            puts("***----------------***\n");
            puts("Converting position variable to transformation matrix representation\n");
            printf("\n Tocka X=%ld,Y=%ld,Z=%ld,RX=%ld,RY=%ld,RZ=%ld,RE=%ld",
                PoseInfo.val.p.data[0], PoseInfo.val.p.data[1],
                PoseInfo.val.p.data[2], PoseInfo.val.p.data[3],
                PoseInfo.val.p.data[4], PoseInfo.val.p.data[5], PoseInfo.val.p.data[6]);

            FromCoord.x = PoseInfo.val.p.data[0];
            FromCoord.y = PoseInfo.val.p.data[1];
            FromCoord.z = PoseInfo.val.p.data[2];

            FromCoord.rx = PoseInfo.val.p.data[3];
            FromCoord.ry = PoseInfo.val.p.data[4];
            FromCoord.rz = PoseInfo.val.p.data[5];

            FromCoord.ex1 = PoseInfo.val.p.data[6];
            resp1 = ConvertToHTM(&FromCoord, &ToFrame);
            if (resp1 == 0)
            {
                printf("Conversion Successful\n");
            }
        }
        else
        {
            printf("Error with Conversion of position register");
        }
    }
    return 0;
}
```

Slika 48. Kod funkcije *SendPose()*

6.4.2. Pretvorba matrica transformacija u pozicijske varijable

Ova funkcija napravljena je u dvije verzije. Jedna verzija iščitava podatke o matrici transformacija iz *string-a* poslanog preko *Socket* komunikacije, dok druga funkcija iščitava podatke iz registara realnih brojeva samog kontrolera. Svrha prve izvedbe je automatsko primanje informacija sa računala preko mreže i spremanje u nove pozicijske varijable kontrolera. Druga funkcija daje za mogućnost korisniku ručno upisivanje članova matrica

transformacije u *Teach Pendant*, nakon čega se aktivacijom signala 10035 transformiraju i spremaju u pozicijsku varijablu 501.

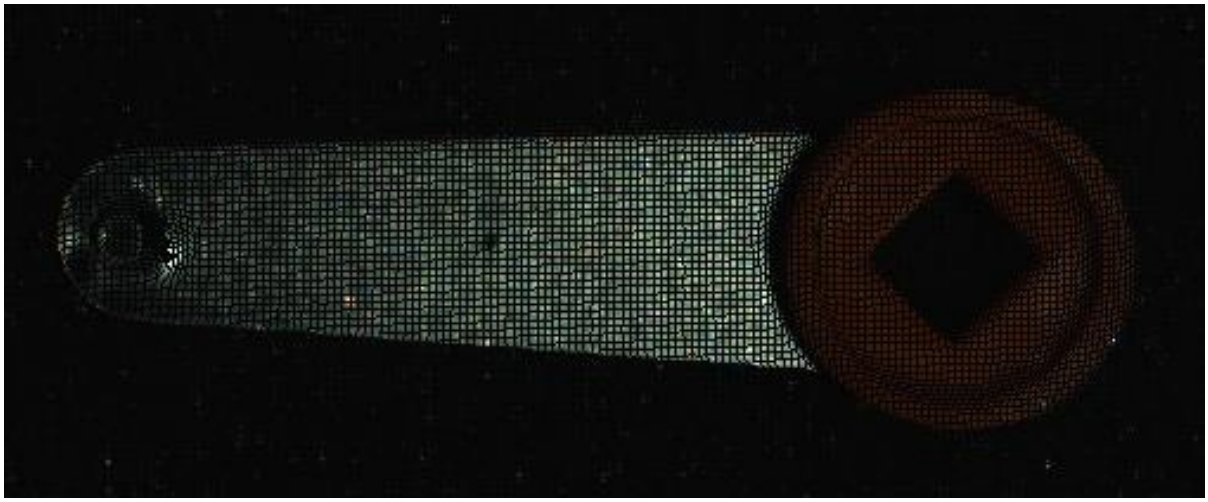
Funkcija transformacije ima sljedeći tok:

1. Čitanje članova matrice transformacije iz registara realnih brojeva pomoću funkcije *mpGetUserVars()* ili ekstrakcija parametara iz *string-a* formata 'n_x, o_x, a_x, p_x, n_y, o_y, a_y, p_y, n_z, o_z, a_z, p_z'.
2. Korištenje funkcije *mpGetUserVars()* za dobavljanje pomoćne pozicijske varijable iz koje se uzimaju podaci o konfiguraciji robota, informacije o vrsti pozicijske varijable, broj alata, korišten koordinatni sustav(u ovom slučaju bazni) i pozicija vanjske osi R_e.
3. Korištenje funkcije *mpFrameToZYXeuler()* koja vrši transformaciju matrice homogenih transformacija u format Eulerovih kuteva i pomaka po x, ,y ,z osima korištenog koordinatnog sustava.
4. Grupiranje i spremanje podataka o poziciji sa podacima pomoćne pozicijske varijable iz koraka 2. pomoću funkcije *mpPutUserVars()*.

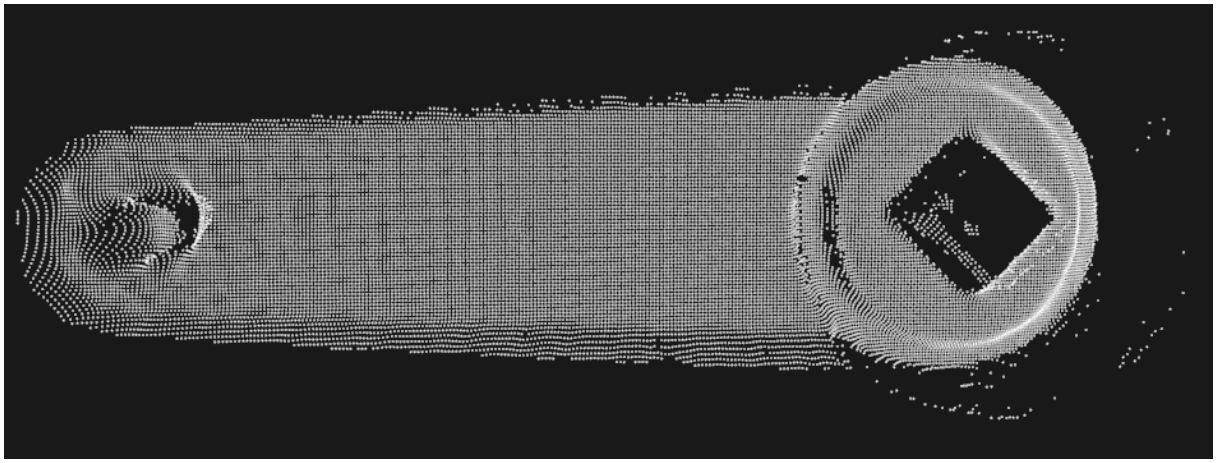
7. Rezultati prepoznavanja

Kao rezultat prikazat će se niz uzetih oblaka točaka, razlike među njima i lokalizirani objekti. Također prikazat će se pozicioniranje robota u odnosu na pronađeni komad na sceni. Za algoritam lokalizacije koriti se *pcl::PointXYZ* format oblaka točaka radi jednostavnosti.

Na Slika 49 vidi se oblak u .zdf formatu dobiven Zivid kamerom, a na slici 50 vidi se oblak točaka pretvoren u *pcl::PointXYZ* format uz filtraciju.



Slika 49. Oblak modela u .zdf formatu



Slika 50. Oblak modela u .pcd formatu

7.1. Utjecaj i kompenzacija okolišnih faktora

Zivid kamera može koristi *Capture()* i *AssistedCapture()* mod rada. Korištenjem *Capture()* moda kamera koristi fiksne parametre zadane u algoritmu. Nasuprot tome, *AssistedCapture()* mod rada koristi način rada u kojem prvotno uzima nekoliko probnih oblaka točaka i prilagođava parametre kamere i filtera kako bi se dobio što kvalitetniji konačni oblak točaka sa što većom gustoćom. Razlika između *Capture()* i *AssistedCapture()* moda rada vidljiva je na

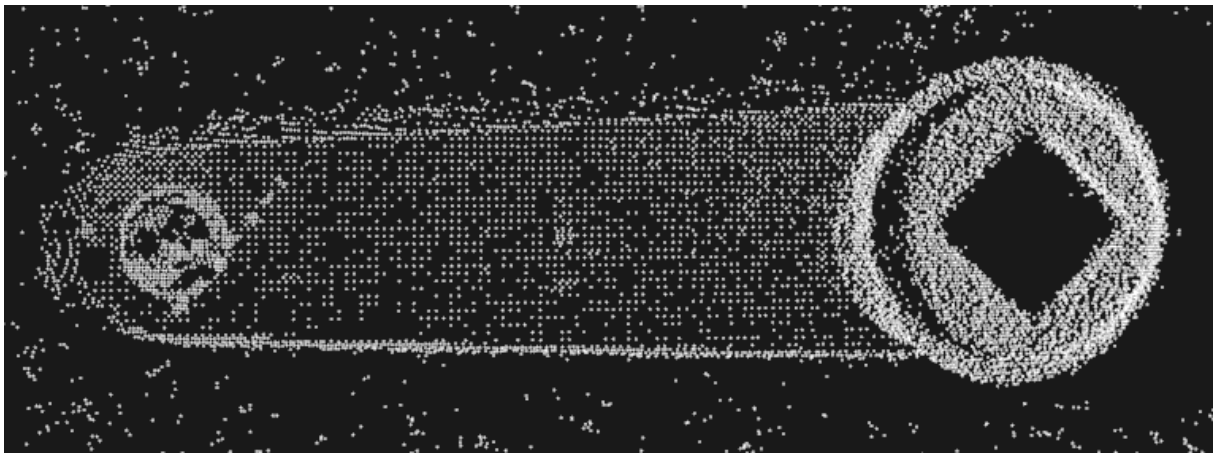
slici 51, gdje je prikazan oblak točaka u .zdf formatu bez prilagođavanja parametara kamere. Dok je na slici 52 prikazan oblak točaka snimljen sa *AssistedCapture()* modom rada.

Slika 51. Oblak točaka uzet sa *Capture()* modom rada 1

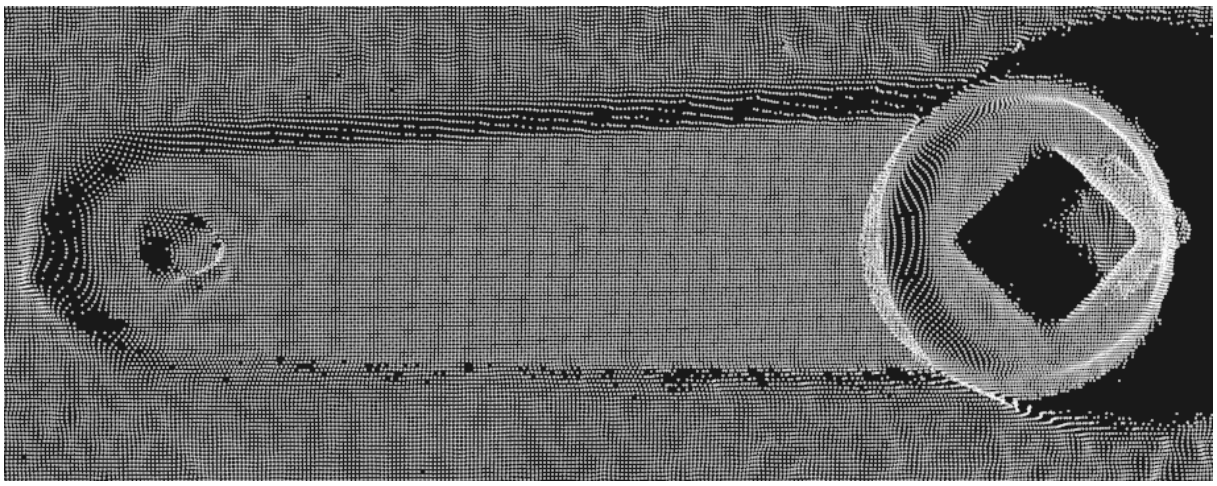
Slika 52. Oblak točaka uzet sa *AssistedCapture()* modom rada 1

Razlika je već vidljiva na prethodne dvije slike (vidi slike 51 i 52) ali kada se pobliže pogledaju oblaci točaka na slikama 53 i 54 očito je da je oblak točaka snimljen sa *AssistedCapture()*

modom rada višestruko gušći od onog snimljenog bez podešavanja parametara. Mogućnost korištenja ovakvog načina rada je od ključnog značaja, budući da za neprilagođene parametre kamere objekt od interesa znatno varira po gustoći točaka ovisno o poziciji i orijentaciji predmeta na sceni, te o vanjskim uvjetima. Manja gustoća oblaka točaka će neizbježno rezultirati lošijim normalama, deskriptorima i korespondencijama a samim time i lošijom lokalizacijom, ako algoritam uopće konvergira prema točnom rješenju.



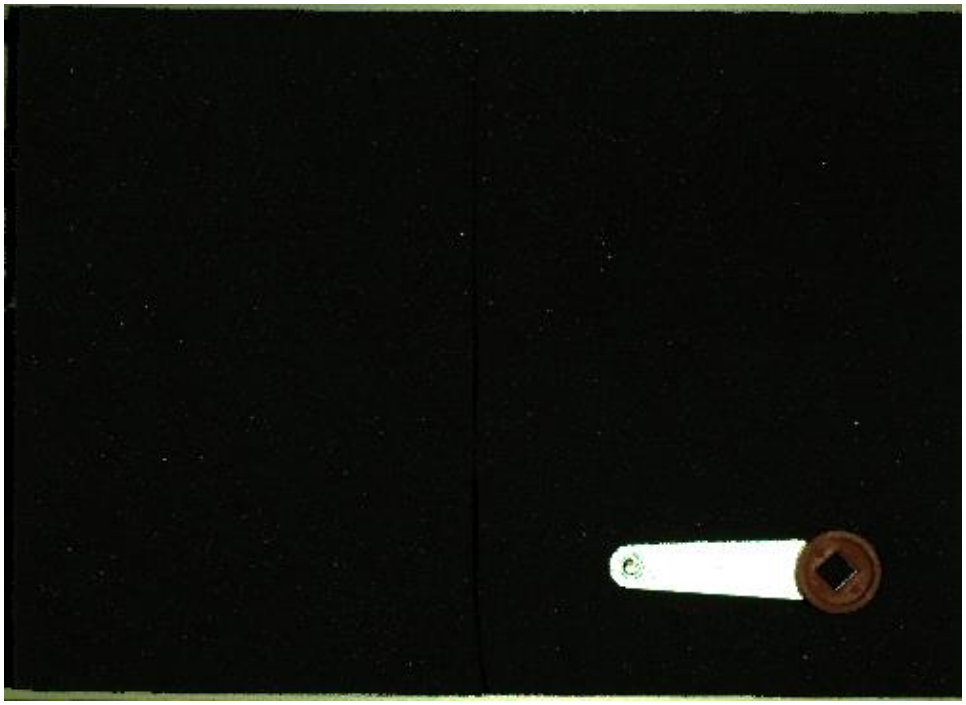
Slika 53. Oblak točaka uzet sa *Capture()* modom rada 2



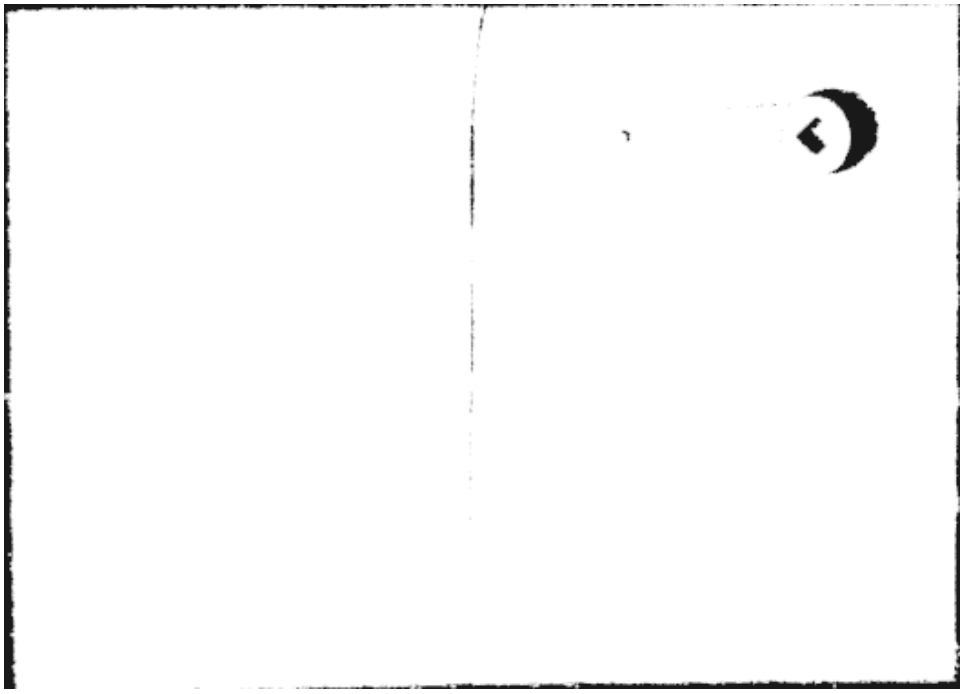
Slika 54. Oblak točaka uzet sa *AssistedCapture()* modom rada 2

7.2. Pretvorba iz originalnog Zivid oblaka točaka u oblak korišten za pretraživanje

Na slici ispod (vidi sliku 55) prikazana je scena oblaka točaka u .zdf formatu, ispod nje nalazi se druga slika u *pcl::PointXYZ* formatu. Kao što se vidi na slikama 55 i 56, nakon konverzije mijenja se perspektiva gledanja oblaka i dobiva se crno bijeli format.



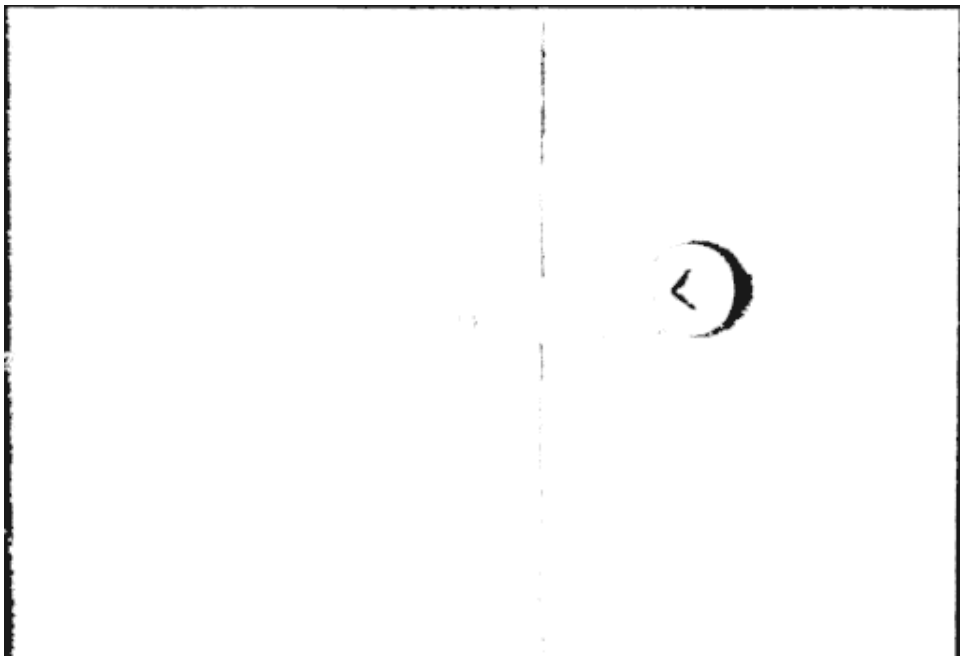
Slika 55. Originalni oblak točaka u .zdf formatu



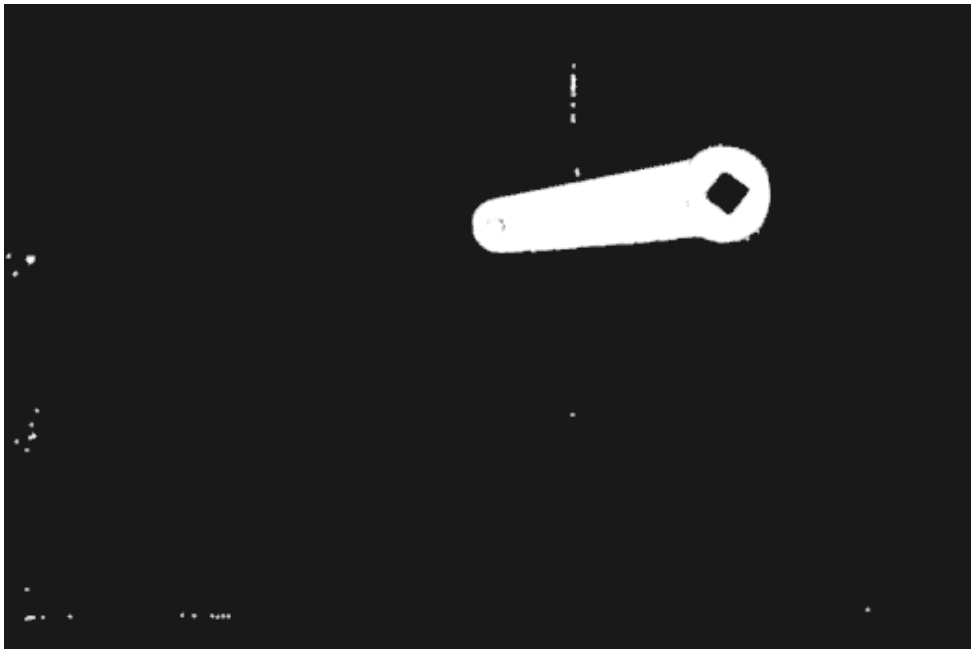
Slika 56. Crno bijeli oblak točaka u .pcd formatu

7.3. Segmentacija ravnine

Da bi se olakšao postupak lokalizacije koristio se filtar za segmentaciju ravnine. Sve točke koje se nalaze 2 mm ili manje od ravnine su uklonjene sa scene. Prikaz scene sa ravninom može se vidjeti na slici 57, prikaz istog tog oblaka točaka nakon uklanjanja ravnine vidljiv je na slici 58. Ovaj postupak će znatno ubrzati lokalizaciju i ukloniti mogućnost loših poklapanja.



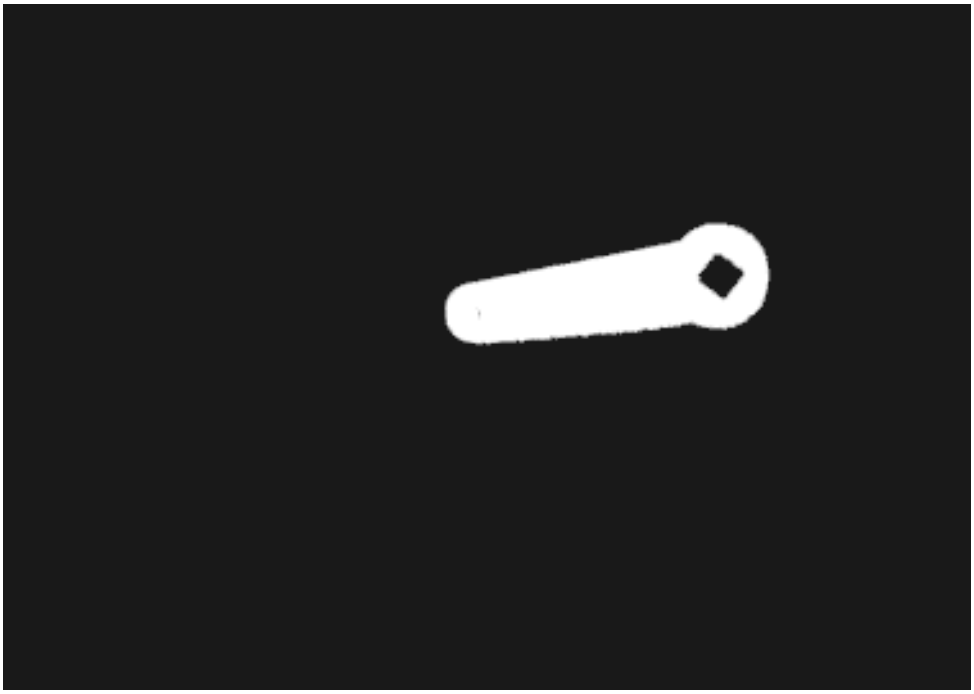
Slika 57. Oblak točaka poslije *PassThrough* filtra i prije segmentacije ravnine



Slika 58. Oblak točaka poslije segmentacije ravnine

7.4. *Statistical Outlier Removal* filter

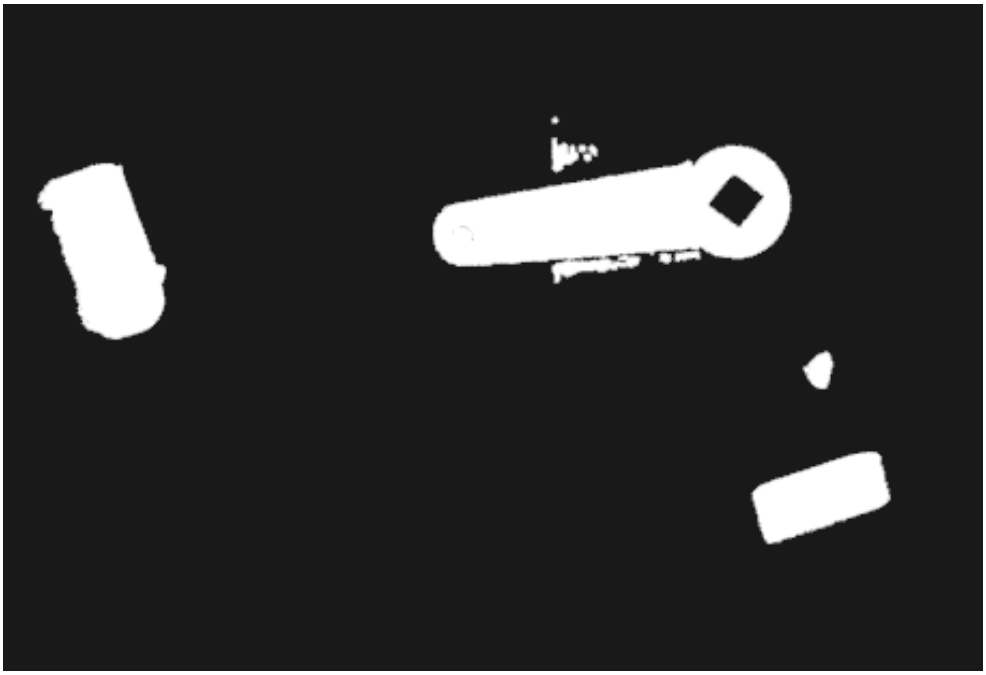
Da bi se uklonili zaostali šumovi, koristi se *Statistical Outlier Removal* filter. Nakon primjene na oblak točaka sa slike 58, dobiva se za rezultat oblak točaka sa slike 59. Zaostali šum je uklonjen.



Slika 59.Oblak nakon koraka na slici 55. i *Statistical Outlier Removal* filtra

7.5. Euklidska klasterizacija

Na slici 60 prikazan je primjer scene gdje se nalaze neželjeni objekti pored objekta od interesa. Da bi se doskočilo tom problemu korišten je postupak euklidske klasterizacije. Nakon korištenja ovog postupka dobiva se oblak točaka prikazan na slici 61. Nepotrebni oblaci su uklonjeni čime se smanjuje mogućnost loših poklapanja te se smanjuje vrijeme lokalizacije.



Slika 60. Oblak točaka prije euklidske klasterizacije

Slika 61. Oblak točaka nakon euklidske klasterizacije

7.6. Pronalaženje komada na sceni(Lokalizacija)

Postupak lokalizacije prikazan je na nekoliko različitih scenarija kako bi se provjerila i prikazala robusnost sustava na različite početne uvjete. Provjerit će se slučajevi:

1. Jedan komad na sceni
2. Više komada na sceni
3. Pojedinačni komadi na sceni sa znatnom promjenom rotacije u odnosu na osnovni model

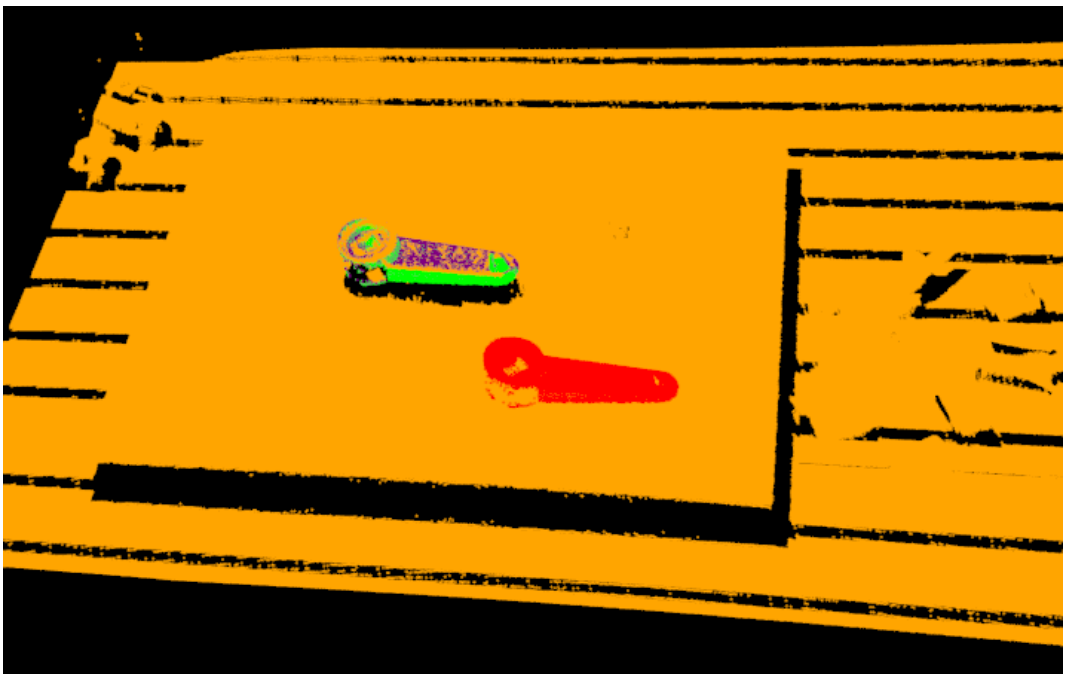
Različiti oblaci točaka prikazani su različitim bojama:

- Oblak crvene boje predstavlja referentni model algoritma lokalizacije, isti taj model umjeren je na robotu
- Oblak zelene boje predstavlja objekt pronađen pomoću Hough-ovog algoritma za klasterizaciju
- Oblak ljubičaste boje predstavlja objekt pronađen pomoću Hough-ovog algoritma za klasterizaciju uz dodatno pozicioniranje oblaka pomoću ICP algoritma
- Oblak narančaste boje predstavlja originalnu scenu koju je slikala kamera bez naknadnih obrada.

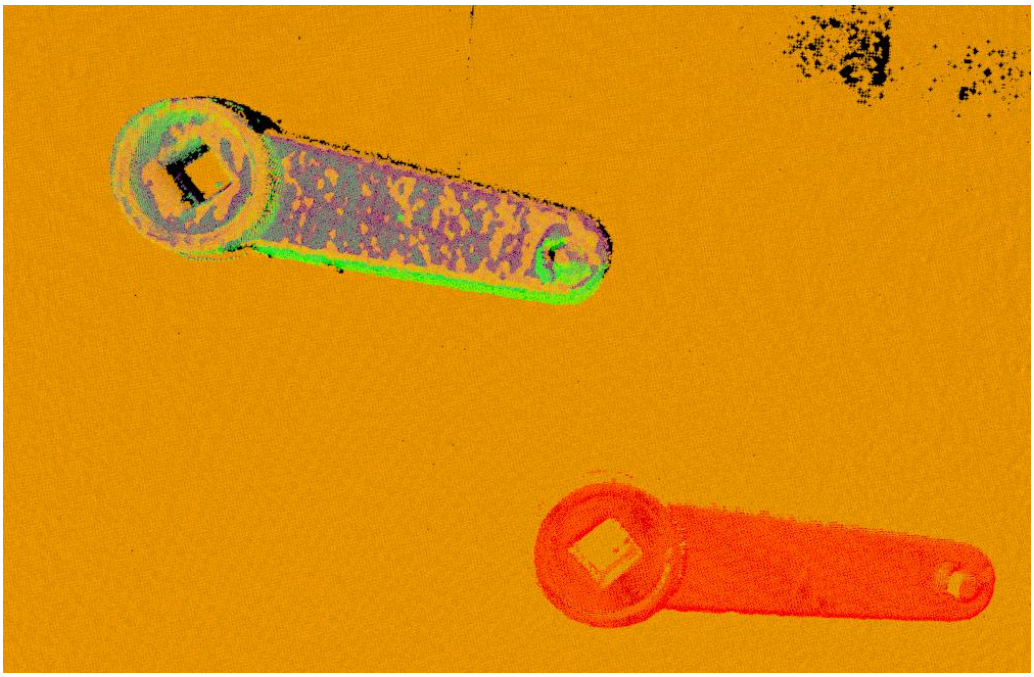
7.6.1. Jedan komad na sceni

Na slici 62 prikazani su rezultati lokalizacije kada se nalazi jedan traženi komad na sceni. Na slici 63 pobliže se može vidjeti utjecaj ICP algoritma, iznos za koji je ICP algoritam ispravio početni oblak točaka može se vidjeti na slici 64 na kojoj prva matrica predstavlja matricu transformacija od referentnog modela do onog pronađenog Hough-ovim algoritmom klasterizacije, dok druga matrica predstavlja dodatnu matricu transformacija kojom se izvršava dodatno preciznije pozicioniranje oblaka točaka.

Matrica koja se šalje robotu preko *Socket* komunikacije vidljiva je na slici 65, ona predstavlja već spomenutu poziciju pronađenog objekta u odnosu na bazni koordinatni sustav robota. Pozicioniranje robota nakon lokalizacije vidljivo je na slici 66 gdje točka na sredini objekta predstavlja traženi centar objekta, na tu točku robot je početno umjeren. Vidi se da nakon lokalizacije robot sa zadovoljavajućom preciznošću pronalazi točku od interesa.



Slika 62. Lokalizacija jednog komada na sceni



Slika 63. Utjecaj ICP algoritma na lokalizaciju

```
   Main Instance :
        Correspondences belonging to this instance: 1024

                |  0.989 -0.123 -0.082 |
        R =     |  0.126  0.991  0.033 |
                |  0.078 -0.043  0.996 |

        t = < 166.412, 46.183, -12.187 >

   Main Instance ICP :
        Correspondences belonging to this instance: 1024
ICP:

                |  1.000  0.022 -0.002 |
        R =     | -0.022  1.000 -0.001 |
                |  0.002  0.002  1.000 |

        t = < 1.810, 3.670, -0.202 >
```

Slika 64. Rezultat algoritma lokalizacije za slučaj jednog komada na sceni

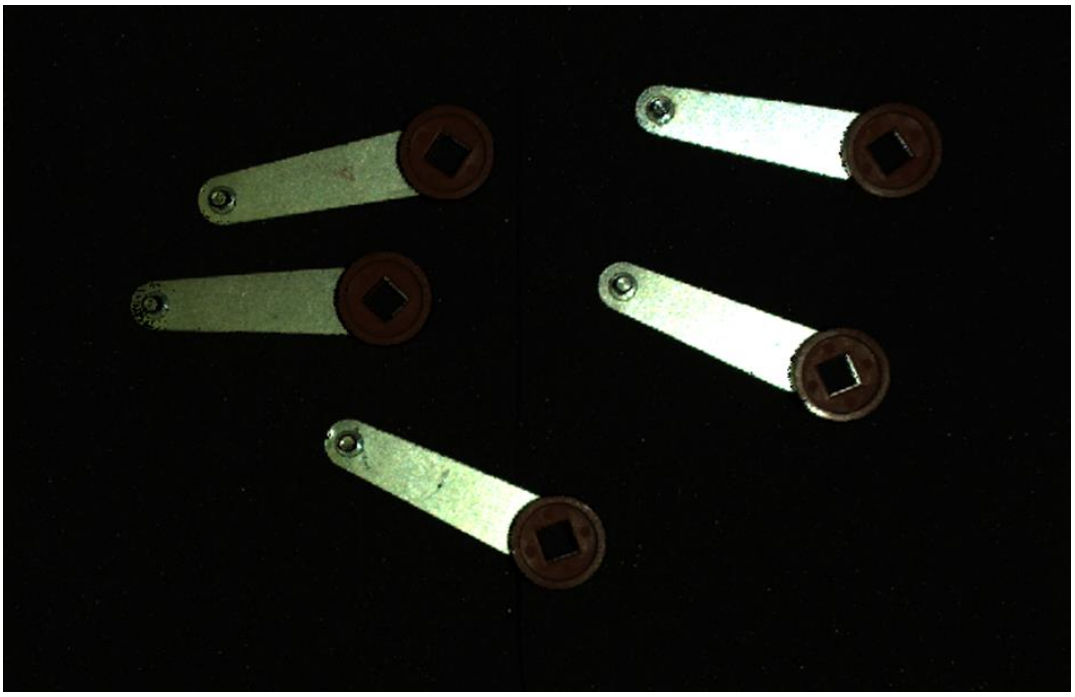
```
Final Displacement matrix for robot:

                | -0.276 -0.960  0.035 |
          R =   | -0.959  0.273 -0.081 |
                |  0.068 -0.056 -0.996 |

          t = < 585.778, -47.975, -429.317 >
```

Slika 65. Matrica za robota, za slučaj jednog komada na sceni

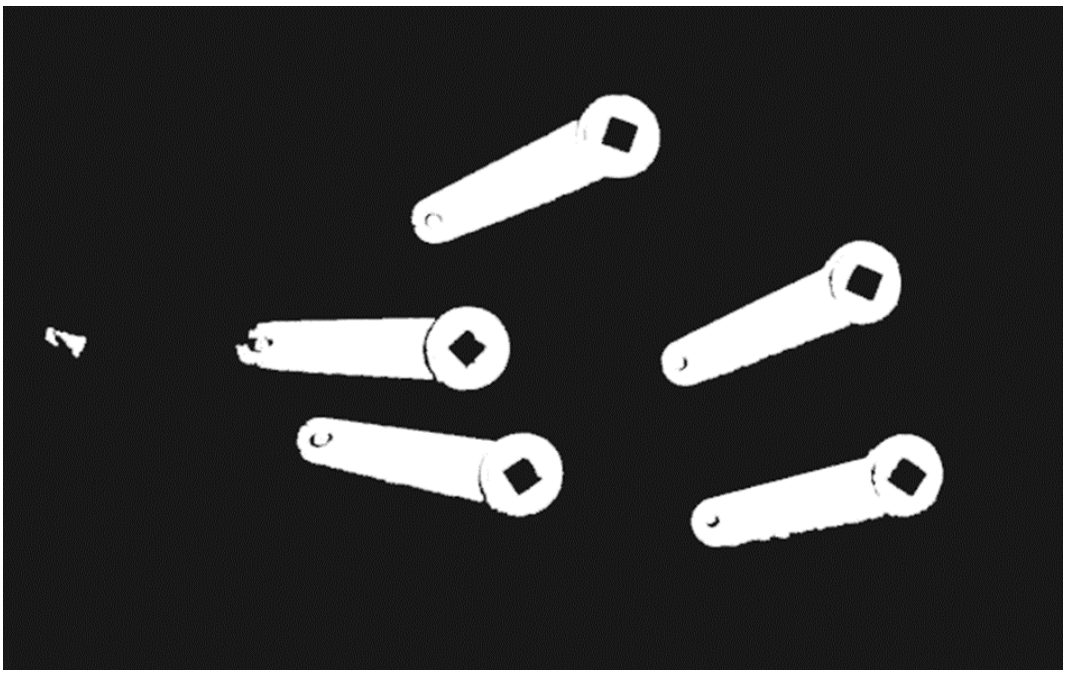
Slika 66. Pozicioniranje robota za slučaj jednog komada na sceni

7.6.2. Više komada na sceni

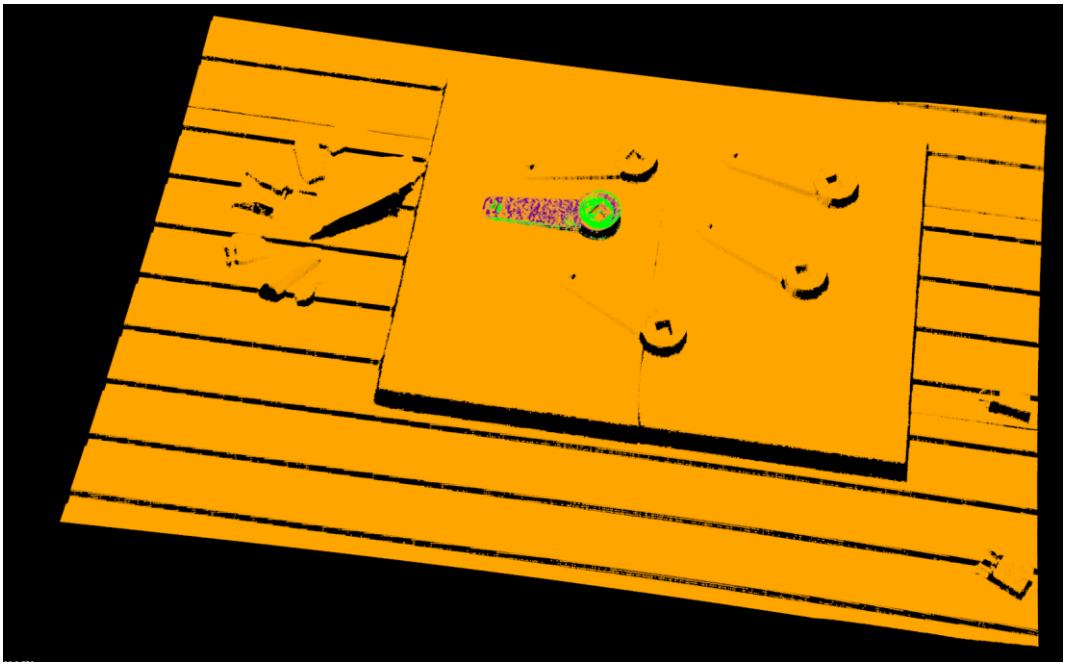
Kod većeg broja komada na sceni(vidi slike 67 i 68) sustav pokušava pronaći sve klastere na sceni, u ovom slučaju 5, budući da je 5 komada na sceni. Nakon toga odabire se klaster sa najvećim brojem korespondencija i na njemu se naknadno provodi ICP algoritam za precizno pozicioniranje. Pronađeni objekt među grupom od više njih na sceni može se vidjeti na slici 69.



Slika 67. Oblak točaka više komada na sceni u .zdf formatu



Slika 68. Oblak točaka više komada na sceni u *pcl::PointXYZ* formatu uz filtriranje

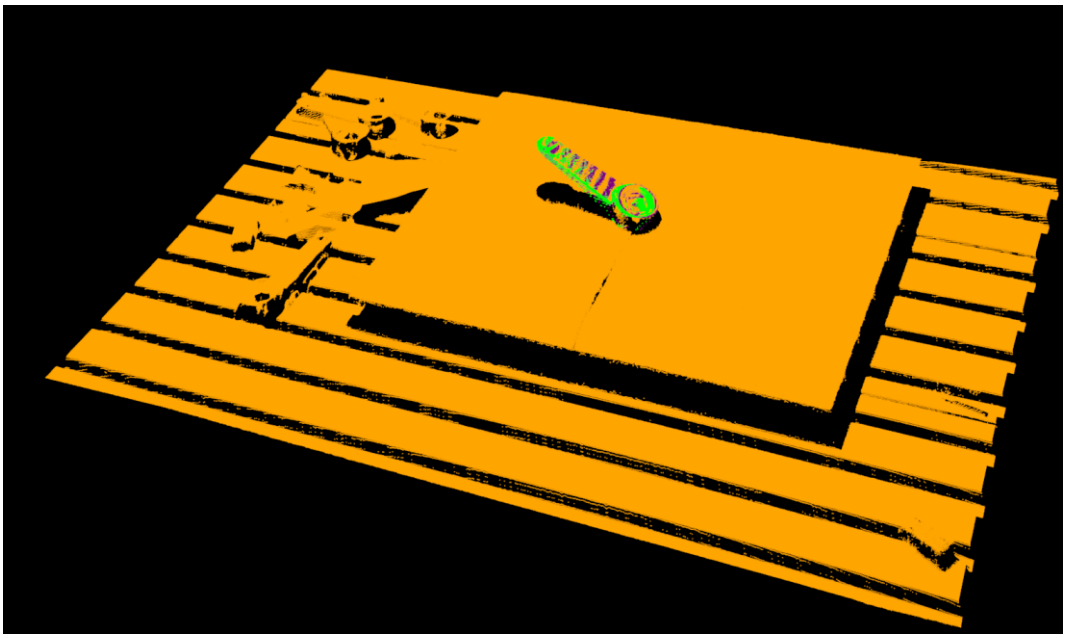


Slika 69. Lokalizacija za slučaj više komada na sceni

7.7. Pojedinačni komadi na sceni sa znatnom promjenom rotacije u odnosu na osnovni model

7.7.1. Primjer 1

Algoritam je također robustan na znatne promjene u rotaciji predmeta što je vidljivo prema slici 70. Točnost lokalizacije je praktički nepromijenjena u odnosu na prethodne primjere. Točnost osigurava dodatna korekcija pozicije oblaka točaka ICP algoritmom koja ima veću ulogu u ovom slučaju što se može vidjeti na slici 71. Matrica transformacija za pozicioniranje robota vidljiva je na slici 72. Također se može vidjeti da manipulator prilagođava orijentaciju alata sukladno orijentaciji pronađenog predmeta(vidi sliku 73).



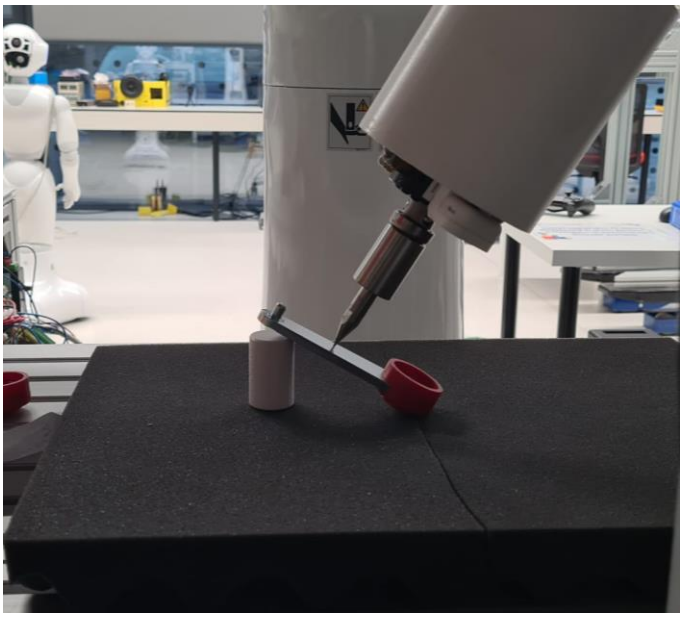
Slika 70. Lokalizacija za slučaj zarotiranog komada 1

```
   Main Instance :
        Correspondences belonging to this instance: 287

             |  0.841 -0.124 -0.526 |
        R =  |  0.118  0.992 -0.045 |
             |  0.528 -0.025  0.849 |

        t = < 530.026, 102.265, 142.507 >

   Main Instance ICP :
        Correspondences belonging to this instance: 287
ICP:

             |  1.000  0.018  0.007 |
        R =  | -0.018  1.000 -0.018 |
             | -0.007  0.017  1.000 |

        t = < -3.425, 16.050, 1.855 >
```

Slika 71. Rezultat algoritma lokalizacije za slučaj zarotiranog komada 1

```
Final Displacement matrix for robot:

             | -0.277 -0.960 -0.048 |
        R =  | -0.814  0.261 -0.520 |
             |  0.511 -0.105 -0.853 |

        t = < 566.447, -102.256, -419.381 >
*----------------*
```

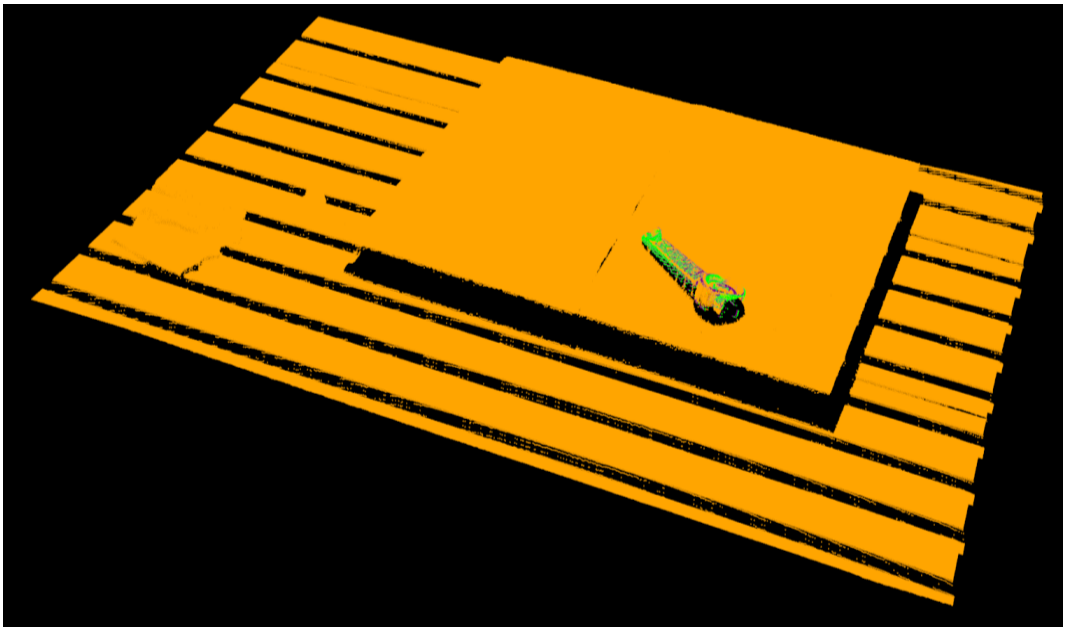
Slika 72. Matrica za pozicioniranje robota, za slučaj zarotiranog komada 1



Slika 73. Pozicioniranje robota za slučaj zarotiranog komada 1

7.7.2. Primjer 2

Na primjeru 2 (vidi sliku 74) može se vidjeti druga uspješna lokalizacija rotiranog predmeta sa pripadajućim matricama transformacija(vidi slike 75 i 76). Pozicioniranje robota u odnosu na pronađeni komad vidljivo je na slici 77.



Slika 74. Lokalizacija za slučaj zarotiranog komada 2

```
  Main Instance :
       Correspondences belonging to this instance: 482

            |  0.826 -0.510 -0.241 |
       R =  |  0.564  0.756  0.332 |
            |  0.012 -0.410  0.912 |

       t = < 322.594, -178.872, 46.930 >

  Main Instance ICP :
       Correspondences belonging to this instance: 482
ICP:

            |  1.000 -0.018  0.010 |
       R =  |  0.018  1.000  0.001 |
            | -0.010 -0.001  1.000 |

       t = < -9.695, -3.630, 1.249 >
```

Slika 75. Rezultat algoritma lokalizacije za slučaj zarotiranog komada 2

```
Final Displacement matrix for robot:

            | -0.693 -0.633  0.344 |
        R = | -0.716  0.661 -0.226 |
            | -0.084 -0.402 -0.912 |

        t = < 647.683, -0.545, -444.563 >
*----------------*
```

Slika 76. Matrica za pozicioniranje robota, za slučaj zarotiranog komada 2

Slika 77. Pozicioniranje robota za slučaj rotiranog komada 2

7.8. Zaprimanje matrica transformacija i prikaz na Telnet-u

Na slici 78 može se vidjeti primjer ispisa članova matrica transformacije na Motoplus aplikaciji nakon iščitavanja članova iz *string-a* dobivenom *Socket* komunikacijom. Na slici 79 može se vidjeti prikaz pozicijske varijable koja je spremljena u pozicijski registar nakon pretvorbe u Eulerovu formu.

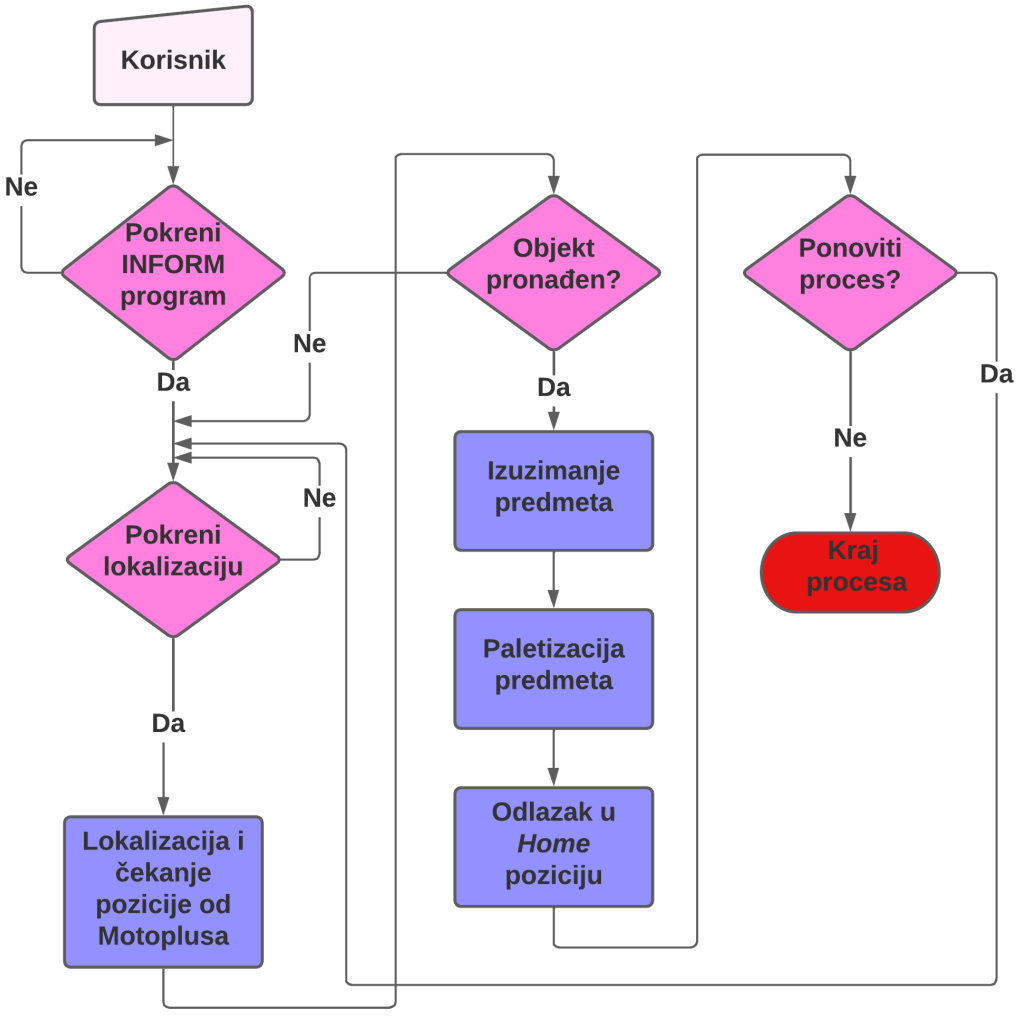
```
Members of transformation matrix are:
-->-0.693440
-->-0.633349
-->0.343557
-->647683.288574
-->-0.715602
-->0.661083
-->-0.225637
-->-545.234680
-->-0.084212
-->-0.402307
-->-0.911632
-->-444563.415527
***----------------***
```

Slika 78. Članovi matrice transformacije za robota prikazani na TELNET sučelju

Slika 79. Prikaz pozicijske varijable robota na Telnet sučelju

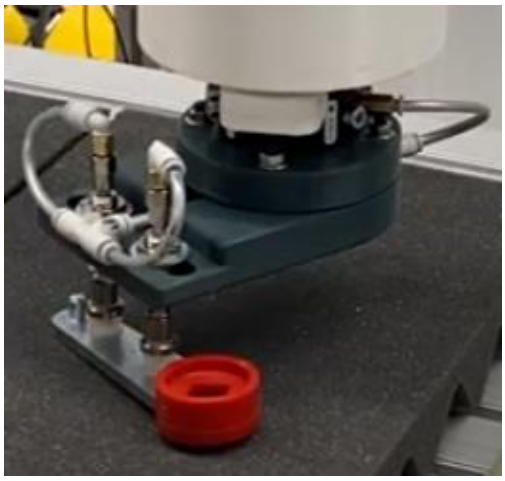
8. *Bin Picking* aplikacija

Bin Picking aplikacija je izvedena pomoću prethodno navedenog sustava lokalizacije. Korištena je prihvatnica iz podpoglavlja 5.4. Za izradu aplikacije korišten je INFORM programski jezik Yaskawa robota naveden u podpoglavlju 4.1.3, pomoću kojeg je napravljen program za upravljanje robotom, istovremeno u pozadini radi Motoplus aplikacija navedena u podpoglavlju 6.4. koja pomoću univerzalnih signala komunicira sa INFORM programom. Korisnik pokreće program u automatskom modu rada robota gdje pomoću *Boolean* varijable može pokrenuti svaki ciklus zasebno. Dijagram toka *Bin Picking* aplikacije prikazan je na slici 80.

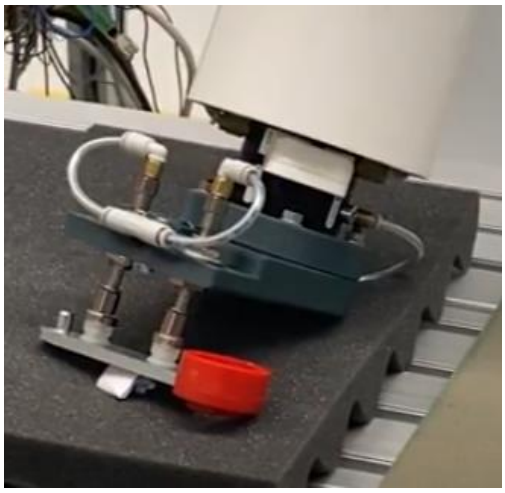


Slika 80. Dijagram toka *Bin Picking* aplikacije

Prikaz hvatanja komada vakuumskom prihvatnicom može se vidjeti na slikama 81 i 82.



Slika 81. Hvatanje komada 1



Slika 82. Hvatanje komada 2

9. Zaključak

U ovom radu predstavljeno je rješenje za lokalizaciju predmeta u nestrukturiranoj okolini. Odrađena je integracija na stvarnom robotu i korišteni su algoritmi izrađeni pomoću PCL biblioteke. Sustav pokazuje zadovoljavajuću točnost lokalizacije i pozicioniranja nakon pronalaska predmeta unatoč svim mogućim greškama koje bi se dodavale jedna na drugu npr. greška pri *Hand Eye* kalibraciji, greške u algoritmu lokalizacije ili greške pri vjernosti oblaka točaka uzetog pomoću Zivid kamere. Dakle, kao prednost sustava može se navesti njegova mogućnost pronalaženja predmeta s prihvatljivom greškom pozicioniranja. Treba naglasiti da ovim radom nije obuhvaćeno mjerenje točnosti pozicioniranja robota ali prema dobivenim rezultatima, vizualno, pretpostavlja se da je greška u submilimetarskom rasponu. Mana sustava je relativno visoko vrijeme izvođenja algoritma za lokalizaciju, to se može kompenzirati na različite načine kao npr. smanjenje gustoće oblaka točaka. Također, treba naglasiti da sustav *Bin Picking-a* u ovom radu ima mjesta za napredak, npr. rješavanje problema planiranja putanje robota za slučaj da bi robot trebao mijenjati konfiguraciju osi tokom izuzimanja komada. Sustav je pogodan za daljnje nadogradnje i modifikacije zbog svoje fleksibilne strukture, mogu se implementirati aplikacije izuzimanja i lokalizacije predmeta sa relativnom jednostavnošću, što zasigurno ostavlja prostora za razvoj u budućnosti.

LITERATURA

[1] M. Crneković, A. Jokić, Predavanja iz kolegija Industrijska robotika, Fakultet strojarstva i brodogradnje, Zagreb, 2021

[2] https://www.zivid.com/zivid-one-plus, Zadnje pristupljeno: Kolovoz 2021.

[3] Solving Pick and Place Automation Challenges with industrial 3D Machine Vision

[4] https://wiredworkers.io/product/zivid-one-plus-medium/, Zadnje pristupljeno: Kolovoz 2021.

[5] Zivid One^{+} Technical specification, Ver.1.0, Zivid AS,Gjerdrums vei 10A,N0484 Oslo, Norway

[6] Zivid One^{+} User Guide, Ver.1.1, Zivid AS,Gjerdrums vei 10A,N0484 Oslo, Norway

[7] https://github.com/zivid/zivid-cpp-samples, Zadnje pristupljeno: Kolovoz 2021.

[8] 3D hand-eye calibration for logistics automation, 06/20, Zivid AS,Gjerdrums vei 10A,N0484 Oslo, Norway

[9] 3D is here: Point Cloud Library (PCL) Radu Bogdan Rusu and Steve Cousins Willow Garage 68 Willow Rd., Menlo Park, CA 94025, USA

[10] Semantic 3D Object Maps for Everyday Robot Manipulation, Radu Bogdan Rusu, Open Perception, Inc., San Francisco, CA, USA

[11] https://pcl.readthedocs.io/projects/tutorials/en/latest/correspondence_grouping.html#correspondence-grouping, Zadnje pristupljeno: Kolovoz 2021.

[12] Tombari, F., Salti, S., & Di Stefano, L. (2010). Unique Signatures of Histograms for Local Surface Description. Lecture Notes in Computer Science, 356–369. doi:10.1007/978-3-642-15558-1_26

[13] A-05-2017, A-No. 163287, YASKAWA Europe GmbH

[14] FS100 Data Sheet, Yaskawa America, Inc | Motoman Robotics Division, March 2017

[15] YR-FS100, A-03-2017, A-No. 157077, YASKAWA Europe GmbH

[16] https://www.motoman.com/getmedia/60fcee9a-b781-46c5-9bd5-70644fb53abf/MotoPlusSDK.pdf.aspx, Zadnje pristupljeno: Veljača 2022.

[17] Product Information Quick change system SWS 011, SCHUNK GmbH & Co.KG Spann- und Greiftechnik

[18] E. Cicvarić, Robotsko rukovanje predmetima rada u nestrukturiranoj radnoj okolini, Zagreb: Fakultet strojarstva i brodogradnje; 2018.

PRILOZI

I. CD-R disk

II. https://github.com/FSBRepositoryCutura/DiplomskiCutura