Praćenje korijena zavara robotskim vidom

Marinović, Ilija

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Ilija Marinović

Zagreb, 2019.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentori:

Prof. dr. sc. Bojan Jerbić, dipl. ing.

Student:

Ilija Marinović

Zagreb, 2019.

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

Zahvaljujem se prof. dr. sc. Bojanu Jerbiću što mi je bio mentor i što mi je omogućio izradu završnog rada u području robotike te na velikom strpljenju, uloženom trudu i pomoći pri izradi završnog rada. Zahvaljujem se i dr. sc. Filipu Šuligoju na konstruktivnim kritikama i pomoći pri izradi diplomskog rada. Zahvaljujem se svim djelatnicima u zavodu za robotiku i automatizaciju proizvodnih sustava na pomoći prilikom izrade diplomskog rada.

Posebna zahvala firmi Visor d.o.o i svim njenim zaposlenicima na podršci i strpljenju te ustupanju određene opreme korištene u izradi ovog rada.

Zahvaljujem se roditeljima, obitelji i prijateljima na strpljenju i podršci tijekom studija .

Ilija Marinović

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:	
Ur. broj:	

DIPLOMSKI ZADATAK

Student: **ILIJA MARINOVIĆ** Mat. br.: 0035193433

Naslov rada na hrvatskom jeziku: **Praćenje korijena zavara robotskim vidom**

Naslov rada na engleskom jeziku: **Tracking welding seams with robotic vision**

Opis zadatka:

U radu je potrebno istražiti mogućnost primjene 2D vizijskog sustava u procesu robotiziranog praćenja različitih geometrijskih profila zavara. Na snimci inicijalnog predmeta rada, dobivenoj pomoću industrijske kamere, potrebno je označiti konturu zavara. Nakon početnog učenja trajektorije, pomoću kamere i linijskog laserskog projektora robotski sustav mora autonomno pratiti zavar te korigirati putanju u skladu s varijacijama predmeta rada. Sličnim predmetima se u ovom slučaju smatraju oni koji u grubom procesu metaloprerađivačke izrade ne odstupaju više od 10mm od inicijalnog predmeta rada.

Za rješavanje zadatka potrebno je:

- izraditi sučelje za unos podataka
- primijeniti i prilagoditi algoritme strojnog vida (OpenCV knjižnica) za otkrivanje kontura zavara
- povezati i kalibrirati koordinatne sustave robota, kamera i lasera
- ostvariti komunikaciju računala s industrijskim robotom i omogućiti kontinuirani prijenos točaka koje opisuju predmet iz koordinatnog sustava vizijskog sustava u koordinatni sustav robota
- razviti neprekinuto upravljanje robotom i praćenje konture zavara s točkom koja opisuje vrh alata robota
- izmjeriti odstupanja i očekivanu točnost kod praćenja konture zavara pomoću robotskog alata.

Rješenje je potrebno implementirati na postojećoj opremi koja je raspoloživa u Laboratoriju za projektiranje izradbenih i montažnih sustava: UR robot, Basler industrijske kamere i linijski laserski projektor.

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

Zadatak zadan:
02. svibnja 2019.

Rok predaje rada:
04. srpnja 2019.

Predviđeni datum obrane:
10. srpnja 2019.
11. srpnja 2019.
12. srpnja 2019.

Zadatak zadao:

prof. dr. sc. Bojan Jerbić

Predsjednica Povjerenstva:

prof. dr. sc. Biserka Runje

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

POPIS TEHNIČKE DOKUMENTACIJE

1 calibration_tool

POPIS OZNAKA

Oznaka	Jedinica	Opis
φ	rad	Fazni pomak između emitiranog i primljenog signala
θ	-	Trodimenzionalni vektor rotacije
∅	rad	Kut skretanja
θ	rad	Kut nagiba
ψ	rad	Kut valjanja
e	-	Jedinični vektor
R	-	Matrica rotacije
A	mm	Duljina duže laserske linije
B	mm	Duljina kraće laserske linije

SAŽETAK

U ovom radu su razvijena i implementirana programska rješenja koja služe kao polu-automatski sustav za praćenje konture zavara. Rad se bavi rješavanjem problema korekcije putanje vrha alata na temelju slike konture zavara na kojoj se vidi laserska zraka. Program i algoritam za praćenje su razvijeni na način da korisniku dozvoljava kompletnu modifikaciju svih elemenata. Za potrebe ovog rada razvijena je komunikacijska veza između robota, kamera i računala. Napravljen je sustav u kojem korisnik odabire željenu trajektoriju zavara koju robot kasnije sam korigira pomoću vizijskih značajki. Interaktivno korisničko sučelje nudi korisniku mogućnost odabira više načina rada, također ostavljene su opcije rada koje omogućuju nove kalibracije i dodatnu modifikaciju programskog koda. Okosnicu rada čini komunikacija koja je razvijena između robota, računala i kamera na način da robot svoje položaje javlja računalu dok s druge strane računalo obrađuje informacije koje dobiva od kamere te ih šalje robotu. Uz to prikazana je kompletna analiza slike procesirana u knjižnici otvorenog koda OpenCV uz pomoć softverskog paketa Visual Studio 2015 u kojem je pisan C++ kod. Obrađeni su i teoretski segmenti povezani s procesiranjem slike i kinematikom robota.

Struktura rada sastoji se od uvoda, osnovnih informacija o korištenim elementima sustava i njihovoj interakciji, razrada analize slika, teoretska podloga iz robotike i vizijskih sustava, povezivanje s robotom te naposljetku slijedi implementacija programa.

Ključne riječi: robotika ; opencv ; ur5 ; praćenje konture ; vizijski sustavi

SUMMARY

This paper deals with development and implementation of programmed solutions that serve as semi-automatic system for tracking the weld seam. The paper deals with solving the problem of correcting the TCP based on the contour image of the disturbance of laser beam. The program and algorithm were developed in a way that allows the complete modification of all the elements. The interactive user interface offers a wide range of possibilities of work and also consists work options that enable new calibrations and additional modification of programmed code. The core of the work is communication developed among the robot, computer and cameras in the way that robot firstly sends its positions to the computer, secondly the computer processes information received from the camera and after that information is sent back to the robot again. Furthermore, the complete image analysis is processed in the open source library OpenCV with the help of the Visual Studio 2015 with the C ++ code written. The theoretical segments connected to the image processing and robot kinematics were processed.

The structure of work is consisted of introduction, basical information about the system element used together with their interaction, complete analysis of the images, Theoretical data from robotics and vision inspection system, connection with the robot and the implementation of the system.

Key words: robotics; opencv; seam tracking; machine vision

1. UVOD

Teško je ne primijetiti brzinu kojom se razvija tehnologija oko nas. Računala koja su u prošlom stoljeću zauzimala čitave prostorije danas svatko nosi u svom džepu. To je sfera industrijske revolucije i napretka koja je zahvatila čovječanstvo pojavom poluvodiča. Međutim i s druge strane, danas u industrijskoj proizvodnji, svjedočimo užurbanom napretku i razvoju novih tehnologija. Najveće zasluge za tu novu četvrtu industrijsku revoluciju idu razvoju pametnih sustava u proizvodnji dobara.

Roboti u proizvodnji su postali normalna pojava još krajem prošlog stoljeća, no ugradnja različitih senzora robotima će dati potpuno novu dimenziju tj. povratnu vezu koju će računalo moći procesirati i unaprijediti. Već je poznato da čovjek nije najbolji radnik u monotonim operacijama zbog umora. Također, čovjeka se ne može poslati da obavlja neki posao u opasnoj okolini, za razliku od robota koji zbog svoje ponovljivosti, fleksibilnosti primjenu nalazi i u poslovima za koje se smatralo da su izrazito namijenjeni ljudima. Jedan od takvih sektora u industrijskoj proizvodnji je zavarivanje. Doduše, zbog štetnih plinova koji se otpuštaju prilikom zavarivanja okolina sigurno nije naklonjena čovjeku, ali tehnika zavarivanja je ipak nešto što svaki zavarivač nauči individualno. Roboti za zavarivanje se danas koriste uglavnom u automobilskoj industriji gdje robotske ruke s ekstenzijom za točkasto zavarivanje „napadaju“ automobilske karoserije na traci i zavaruju njihove dijelove, no s druge strane roboti za zavarivanje se koriste i u velikim serijskim proizvodnjama gdje se zavaruje i ostalim metodama kao što su: MIG, MAG, TIG i elektrolučno zavarivanje. U većini slučajeva je geometrija predmeta zavara strogo definirana pa se robot kreće po unaprijed definiranoj putanji, no postoje i slučajevi gdje robot sam programira svoju putanju pomoću senzora kao što su laser i kamera.

Ovaj rad se djelomično bavi tom problematikom, praćenje konture zavara pomoću kamere i lasera. Projekt je napravljen tako da korisnik odabire konturu zavarivanja za jedan radni komad nakon čega robot provodi snimanje laserom i kamerom kako bi se odredila preciznija putanja. Po završetku eventualnog zavarivanja zamišljeno je da se promjeni radni komad za koji nije nužno da bude savršeno obrađen kao prethodni komad jer robot će pomoću kamere i lasera uspjeti kompenzirati pogrešku pozicioniranja od 10mm.

2. ELEMENTI SUSTAVA

U samom uvodu je već napisano kako je ovaj rad zamišljen kao tehnički sustav koji integrira robota, dvije kamere, laser i računalo. Računalo je zamišljeno kao jezgra cijelog sustava te su svi elementi sustava preko jednog preklopnika spojeni na računalo koje preko aplikacije upravlja njima.

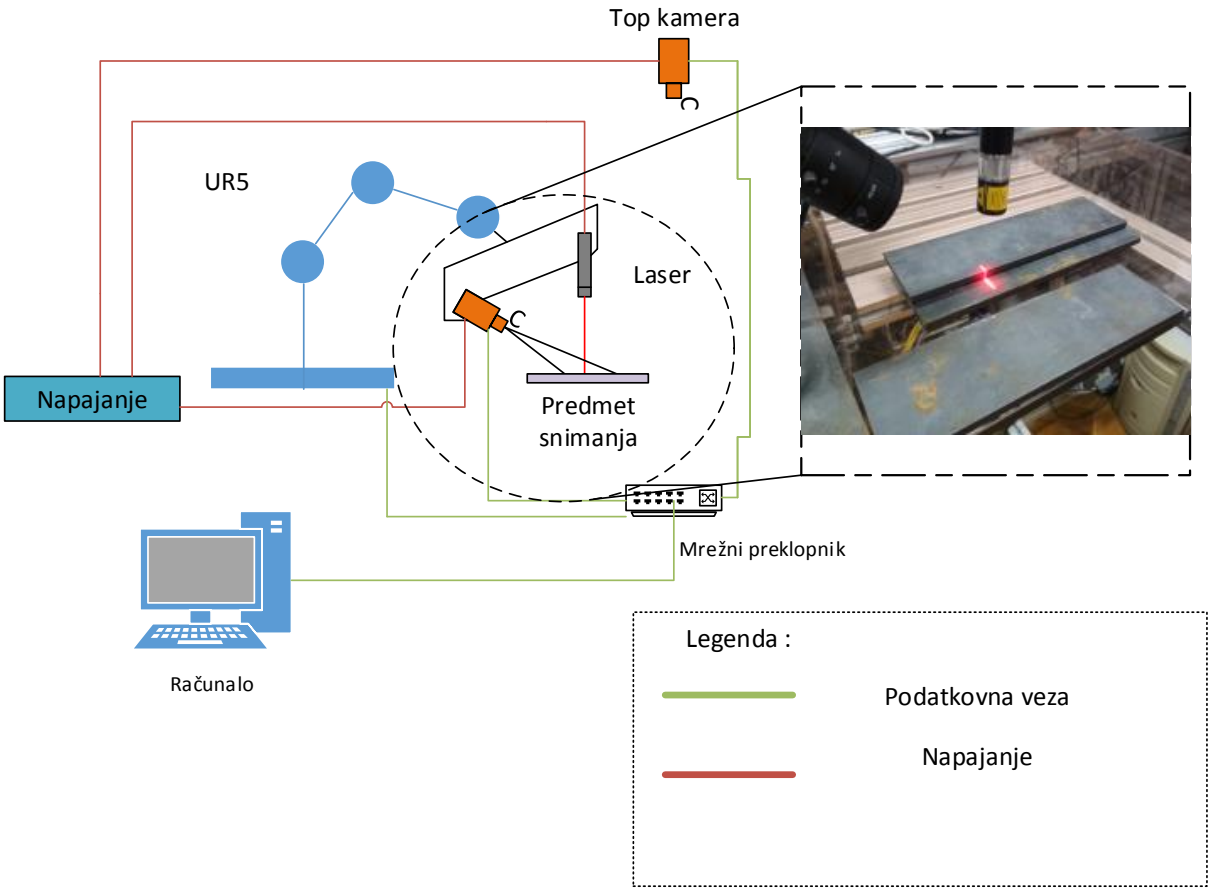
Kako bi se bolje razumio cijeli sustav nadalje će u ovom radu biti posebno objašnjeni svi hardverski i softverski elementi sustava.

Hardverski elementi sustava:

- Industrijska robotska ruka: UR5
- Industrijska kamera koja gleda konturu zavara: Basler GigE AC3800-10gc
- Gornja industrijska kamera: Basler GigE AC1800-30gm
- Računalo
- Mrežni preklopnik
- Industrijski laser : ZLaser ZM18S

Softverski elementi sustava:

- Microsoft Visual studio
- OpenCV
- Pylon Viewer

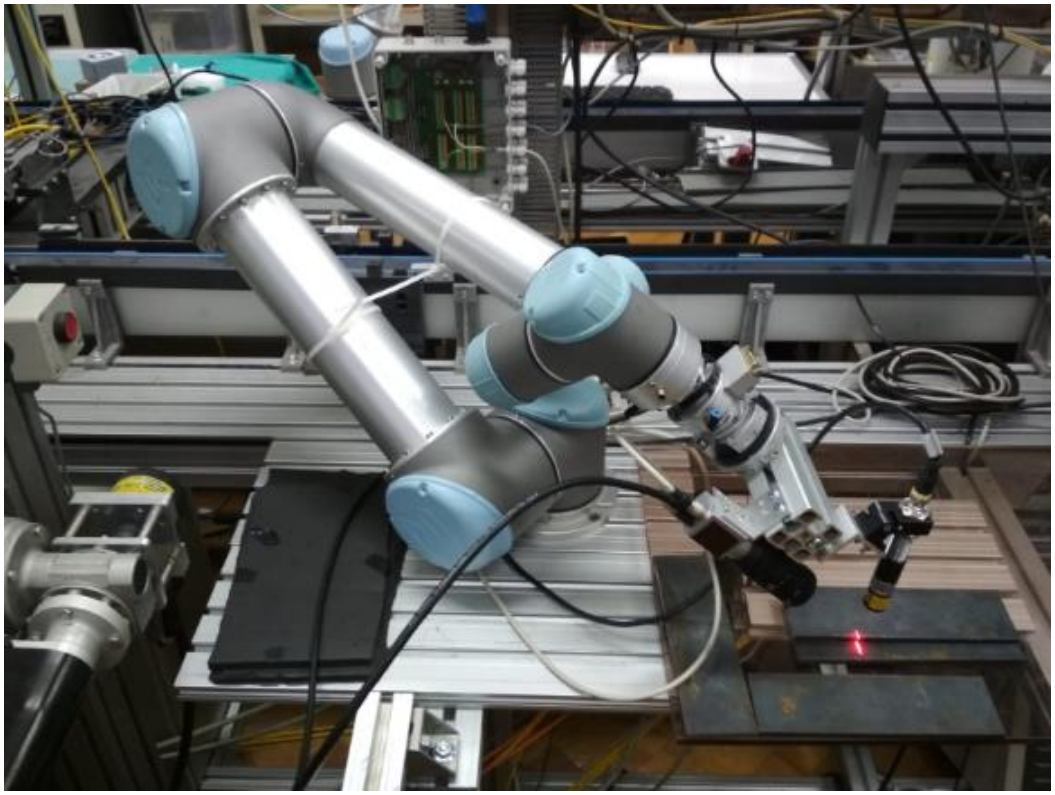


Slika 1:Shematski prikaz sustava

2.1 HARDVERSKI ELEMENTI SUSTAVA

2.1.1 Robot

Robot UR5 prikazan na Slici 2 je reprogramabilni višenamjenski manipulator danskog proizvođača *Universal Robots* čije su osnovne specifikacije dane u tablici 1. Na njegovu prirubnicu postavljen je aluminijski profil koji nosi kameru i laser za snimanje zavara. Robotu je dodijeljena mrežna IP adresa *192.168.0.60* te je fizički UTP kabelom spojen na preklopnik. Ethernet komunikacija (*socket messaging*) omogućava slanje skriptnih komandi preko računalnog programa napisanog u C++ programskom jeziku na kontroler robota. Primjer takvog koda kao i neke funkcije bit će objašnjeni kasnije u ovom radu.



Slika 2:Robot UR5

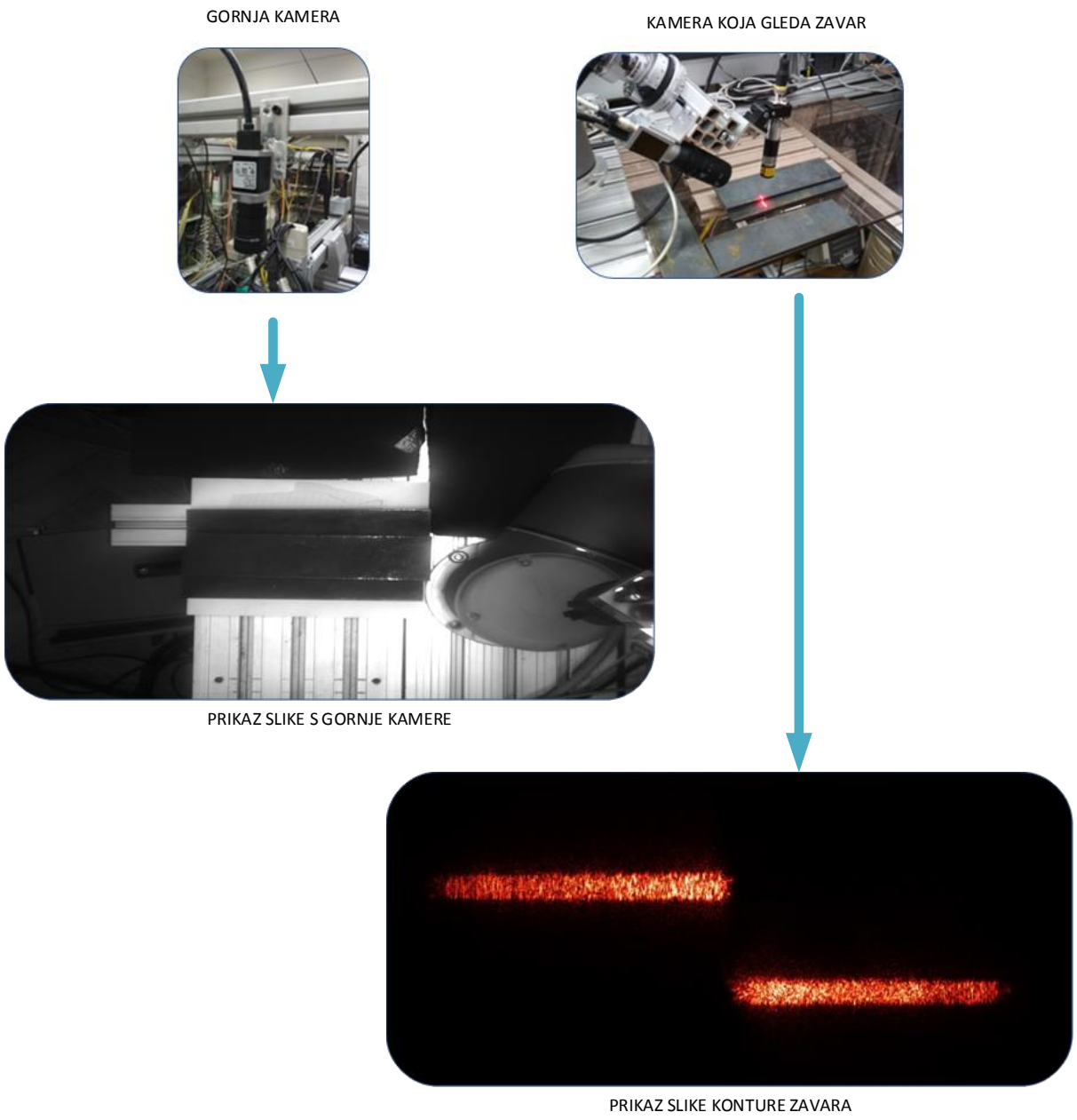
Tablica 1: Tehničke specifikacije UR5robota [1]

Masa:	18,4 kg
Nosivost:	5 kg
Doseg:	850 mm
Opseg zglobova:	+/- 360°
Brzina svih zglobova:	180 °/s
Brzina alata:	1 m/s
Ponovljivost:	+/- 1 mm
Promjer baze:	149 mm
Broj stupnjeva slobode:	6
Veličina upravljačke jedinice:	475 x 423 x 268 mm
Programiranje:	Polyscop grafičko sučelje
Potrošnja energije:	~200W
Materijali:	Aluminij i plastika
Raspon radne temperature:	0 – 50 °C

2.1.2 Kamere

U ovom radu korištene su dvije industrijske kamere njemačkog proizvođača *Basler*. Jedna kamera je postavljena na robotsku ruku te se s njom snimala kontura zavara, dok je druga kamera postavljana iznad radnog stola te je snimala predmet obrade kako bi se odredilo željeno područje zavarivanja.

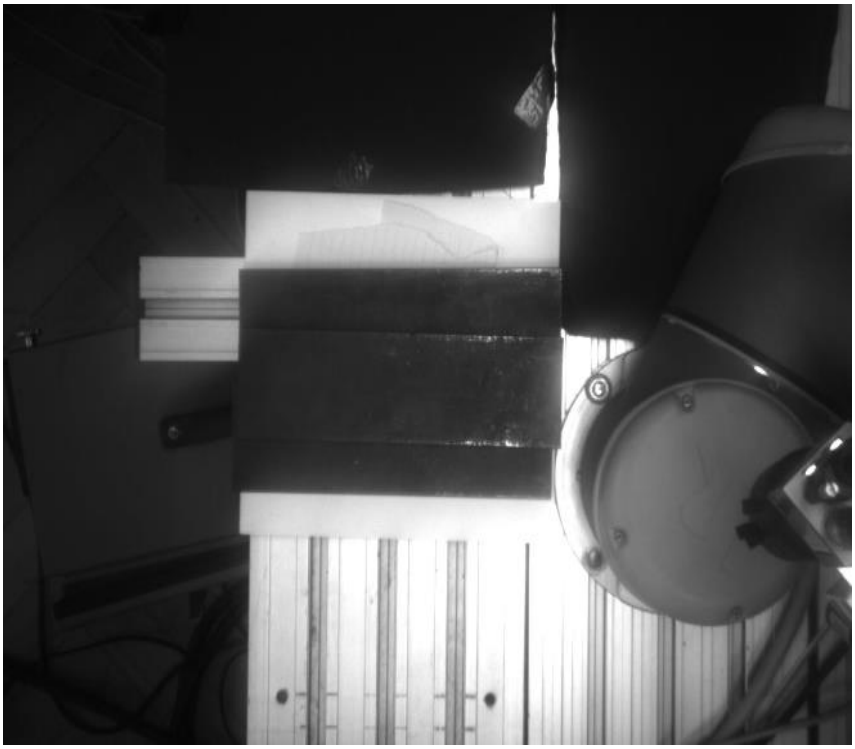
Model kamere koja snima konturu zavara je Basler GigE acA3800-10gc te ima senzor u boji MT9J003 CMOS s kojim razvija 10 slika po sekundi rezolucije 3800x2748 px. S obzirom da je to relativno malo, a u ovom radu nema potrebe za tako velikom rezolucijom, rezolucija je prepolovljena kako bi se dobio duplo veći *frame rate* (slika 3.). Kamera je spojena UTP kabelom na mrežni preklopnik te joj je u Baslerovom programu *pylon IP Configurator* dodijeljena IP adresa 192.168.0.120.



Slika 3: Prikaz kamera i njihovih slika

Kako predmet snimanja, odnosno zavarivanja, može biti od različitog materijala jasno je kako će refleksija lasera biti drugačija, stoga je potrebno prije snimanja postaviti pravilnu ekspoziciju kamere. Ekspozicija je parametar s kojim se definira koliko će se svijetla pustiti u senzor kamere. Slika 3.u donjem kutu prikazuje sliku konture zavara prije programske obrade. Početna slika je izuzetno bitna jer o njoj uvelike ovisi danja obrada slike tako da je potrebno naći optimalnu ekspoziciju u kojoj bi se linije lasera dovoljno istaknule, ali da opet laser ne „baca“ previše šuma.

Druga kamera koja je smještena iznad predmeta snimanja je također Basler GigE acA1300-30gm te ima Sony ICX445AL CCD senzor te dohvaća crno bijelu sliku rezolucije 1296 x 966px. Osnovna zadaća te kamere je da korisniku da prikaz radnog stola s predmetom snimanja kako bi korisnik okvirno odredio željeno područje zavarivanja. Visina na kojoj je postavljena kamera je 770mm od radnog stola te je programski potrebno unijeti visinu predmeta snimanja kako bi robot to kompenzirao te uvijek snimao s iste udaljenosti. Jednako kao i kamera koja je snima konturu zavara i ova je kamera spojena UTP kabelom do preklopnika te joj je dodijeljena IP adresa 192.168.0.220.



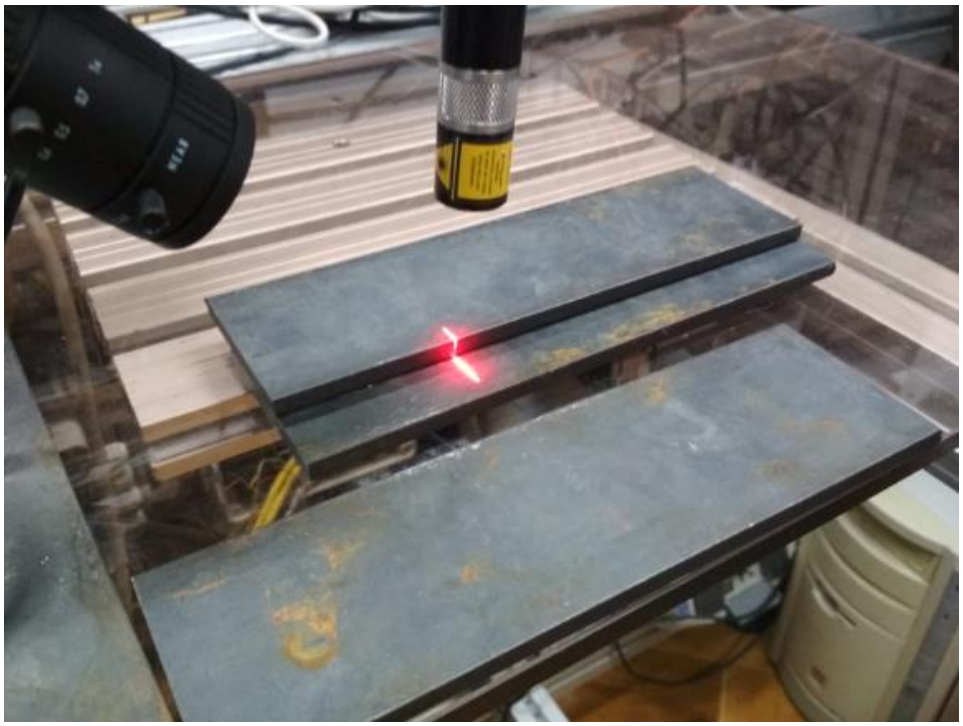
Slika 4:Prikaz slike s Top kamere

2.1.3 Laser

Laser koji se koristio u ovom radu je ZLaser ZM18S valne duljine 640nm s mogućnošću varijabilnog fokusa preko navoja na glavi lasera. Laserska zraka daje liniju crvene boje koja pada okomito na predmet snimanja te se lomi na konturi zavara, što za posljedicu daje prikaz stepenaste linije na slici kamere. Slika 5. prikazuje lasersku zraku na radnom komadu.

Tablica 2:Tehničke specifikacije lasera [2]

Raspon valne duljine	635nm – 830 nm
Maksimalni dopušten napon	30 VDC
Promjer glave	20mm
Radna temperatura	-10°C do 50°C
Utrošak snage	<2W
Radna struja	<400 mA
Zaštita	IP67



Slika 5: Laserska zraka na radnom komadu

2.1.4 Mrežna komunikacija

Svi elementi sustava između kojih je potrebna komunikacija spojeni su na jedan mrežni preklopnik te su im dodijeljene IP adrese na istoj mreži. Tablica 3. prikazuje raspodjelu IP adresa unutra mreže.

Tablica 3:Raspodjela IP adresa

UREĐAJ	IP adresa
Računalo	192.168.0.100
Kamera koja gleda predmet zavarivanja	192.168.0.120
Top kamera	192.168.0.220
Robot	192.168.0.60

2.2 SOFTVER

Kao što je već spomenuto, cijela aplikacija se izvodi na računalu, za što je potrebna neka programska priprema. Cijeli program napisan je u programskom jeziku C++ , dok je za vizijsku analizu slike korištena besplatna knjižnica otvorenog koga *openCV* koja je integrirana u Visual studio.

2.2.1 Visual studio

Slika 6:Visual Studio logo

Visual Studio je integrirano razvojno okruženje (eng. integrated development environment – IDE) distribuirano od Microsofta. Koristi se za razvoj računalnih programa, internet stranica, web aplikacija i mobilnih aplikacija te podržava programske jezike kao što su Python, JavaScript, C#/VB i C++ koji je ujedno korišten u svrhu rada [3].

2.2.2 OpenCV

Slika 7: OpenCV logo

Prema [4], OpenCV (eng. *Open Source Computer Vision Library*) je knjižnica otvorenog koga za razvoj računalnog vida i strojnog učenja. Osnovna zadaća OpenCV-a je skraćivanje vremena potrebnog za vizijsku analizu slike, a to upotpunjuje spoznaja da knjižnica ima više od 2500 optimiziranih algoritama, što uključuje sveobuhvatan skup klasičnih i najmodernijih algoritama računalnog vida i strojnog učenja. Ovi algoritmi mogu se koristiti za otkrivanje i prepoznavanje lica, identifikaciju objekata, klasificiranje ljudskih aktivnosti u videozapisu, praćenje pokreta kamere, praćenje objekata u pokretu, izdvajanje 3D modela objekata, izradu 3D oblaka točaka iz stereo kamera, povezivanje slika kako bi se dobila visoka razlučivost slika cijele scene, pronaći slične slike iz baze podataka slika, itd.

OpenCV ima više od 47 tisuća korisnika i procijenjeni broj preuzimanja je veći od 18 milijuna. Knjižnica se opsežno koristi u tvrtkama, istraživačkim skupinama i osobama koji se ovom granom računarstva bave iz hobija. Izvorno je napisana u programskom jeziku C++ no knjižnica je dostupna i u drugim programskim jezicima kao što su Python, Java i MATLAB,a podržava Windows, Linux, Android i Mac OS .

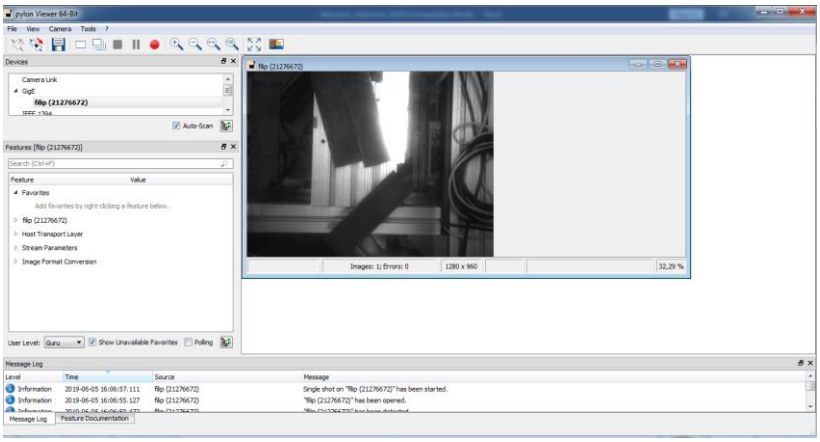
Knjižnicu je potrebno integrirati unutar razvojnog okruženja Visual Studio, nakon čega je moguće pristupiti funkcijama knjižnice. Kasnije u ovom radu će biti prikazanje neke od funkcija iz knjižnice koje su korištene u sklopu ovog rada.

2.2.3 Basler pylon

Slika 8:pylon logo

Prema [5], pylon je softverski je paket koji se sastoji od jednostavnog SDK-a i upravljačkih programa te alata koje možete koristiti za upravljanje bilo kojom Basler kamerom pomoću sustava Windows ili Linux PC ili Mac. Zahvaljujući najnovijoj tehnologiji GenICam 3.1, pylon nudi neograničen pristup najnovijim modelima kamera.

Sučelje za programiranje na pylonu može raditi sa širokim rasponom kamera, omogućujući da se postojeći programski kod koristi bez izmjena za bilo koje Basler kamere i operativne sustave. Knjižnica funkcija je dostupna u raznim programskim jezicima kao što su C, C ++, C #, VB.Net. Sučelje programa je pregledno i jednostavno za korištenje, a prikazano je na slici 10. Upravo zbog grafičkog sučelja i jednostavnosti program odlučno služi za inicijalno traženje parametara snimanja.



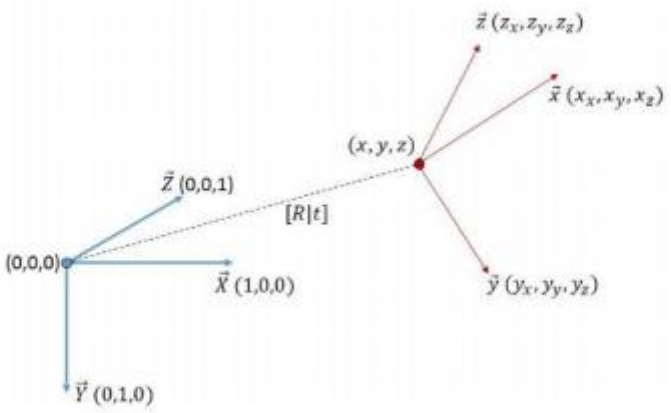
Slika 9: prikaz pylon sučelja

3. PODLOGA ZA UPRAVLJANJE ROBOTOM

U ovom poglavlju će biti prikazane neke matematiče relacije. Uz to proći će se kroz matematiku i pretvorbu koordinatnih sustava kako bi se lakše prikazala problematika razrađena u ovom radu.

3.1 Homogena matrica transformacije

Prema [3], matrica homogenih transformacija sadrži informacije o orijentaciji i translaciji nekog objekta u prostoru. Ona je veličine 3x3 za 2D problem, a 4x4 za 3D problem što je i u ovom radu slučaj. To se može predočiti na vrlo jednostavan način: jedan koordinatni sustav je bazni koordinatni sustav robota, a drugi koordinatni sustav je predmet snimanja sa svojom orijentacijom. Iz tog razloga potrebno je naći matricu transformacije koja prikazuje odnos odnosno poveznicu između ta dva sustava.



Slika 10: Prikaz veze između dva koordinatna sustava [3]

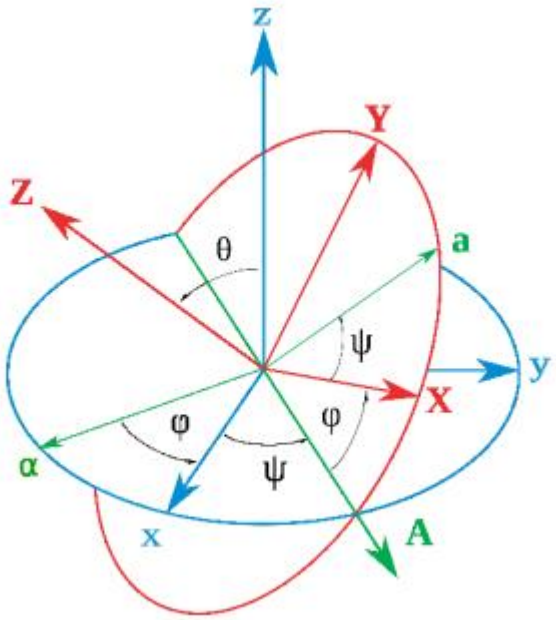
Matrica homogene transformacije za opisivanje koordinatnog sustava na slici 25. je sljedećeg oblika:

$$A = \begin{bmatrix} i_x & j_x & k_x & p_x \\ i_y & j_y & k_y & p_y \\ i_z & j_z & k_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Prva tri stupca i retka u matrici A prikazuju matricu rotacije, dok zadnji stupac prikazuje vektor translacije po x, y i z osi, također treba zamijetiti da je zadnji stupac samo proširenje za 3D sustav te nema nikakve relevantne podatke o rotaciji i translaciji.

3.2 Eulerovi kutovi

Eulerovi kutovi φ, ψ, ϑ, koji određuju relativni položaj jednoga prema drugome dvaju pravokutnih koordinatnih sustava sa zajedničkim ishodištem. Pri tome je φ kut precesije, ψ kut vlastite rotacije, ϑ kut nutacije. Postoji 12 konvencija o prikazu i zapisu eulerovih kutova, a one su : z-x-z, x-y-x, y-z-y, z-y-z, x-z-x, y-x-y, x-y-z, y-z-x, z-x-y, x-t-y, z-y-x, y-x-z.



Slika 11: Prikaz eulerovih kutova

Matrica rotacije je napravljena množenjem matrica Rx, Ry i Rz čiji standardni oblik ovisi o konvenciji, a za x-y-z ona izgleda ovako:

$$Rx = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\varnothing) & -sin(\varnothing) \\ 0 & sin(\varnothing) & cos(\varnothing) \end{pmatrix}$$

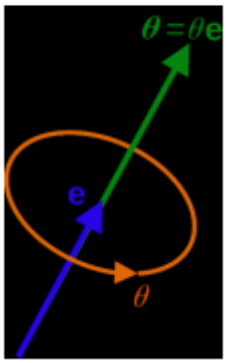
$$Ry = \begin{pmatrix} cos(\varnothing) & 0 & sin(\varnothing) \\ 0 & 1 & 0 \\ -sin(\varnothing) & 0 & cos(\varnothing) \end{pmatrix}$$

$$Rz = \begin{pmatrix} cos(\varnothing) & -sin(\varnothing) & 0 \\ sin(\varnothing) & cos(\varnothing) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

3.2 Axis – angle zapis

Uz eulerove kutove, matricu transformacije i kvaternion postoji još i axis – angle zapis za opisivanje rotacije objekta. Kod axis – angle zapisa rotacija se zapisuje putem trodimenzionalnog vektora $\boldsymbol{\theta}$ koji je skalarni umnožak kuta θ koji predstavlja iznos rotacije i jediničnog vektora $\boldsymbol{e}$ koji pokazuje smjer vrtnje.

$$\boldsymbol{\theta} = \theta e = [R_x\ R_y\ R_z]$$



Slika 12: Axis - angle prikaz

S obzirom da robot UR5 koristi axis – angle zapis potrebno je razjasniti i pretvorbu matrice rotacije u axis-angle zapis.

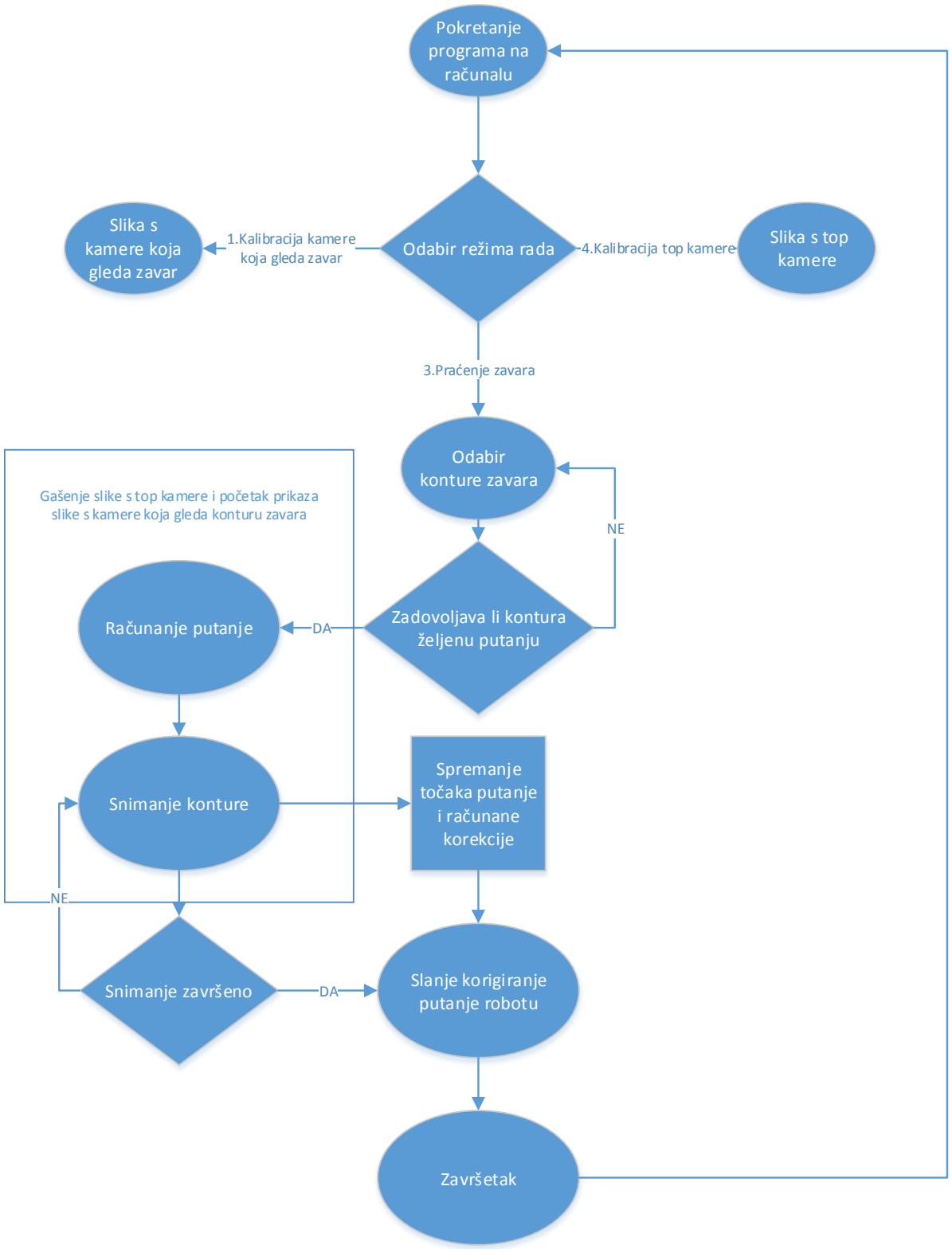
$$\theta = \arccos\left(\frac{1}{2}[R_{11} + R_{22} + R_{33} - 1]\right)$$

$$e_x = \frac{R_{32} - R_{23}}{2sin\theta}$$

$$e_y = \frac{R_{13} - R_{31}}{2sin\theta}$$

$$e_z = \frac{R_{21} - R_{12}}{2sin\theta}$$

4. INTEGRACIJA SUSTAVA ZA DETEKCIJU I PRAĆENJE KONTURE ZAVATA



U ovom poglavlju bit će obrađene sve programske zadaće koje je bilo potrebno obraditi za uspješno provođenje detekcije i praćenja konture zavara. Kroz iduće točke prikazat će se vizijsko procesiranje slike te razvoj i uspostava komunikacije s robotom.

4.1 ANALIZA SLIKE

U ovom poglavlju rada prikazat će se neke od funkcija koje su korištene za analizu slike u ovom radu. Zaključit će se koje su ključne značajke potrebne da bi algoritam dobro radio. Za početak prikazat će se dio funkcija korišten u analizi slike s kamere koja gleda laser odnosno konturu zavara.

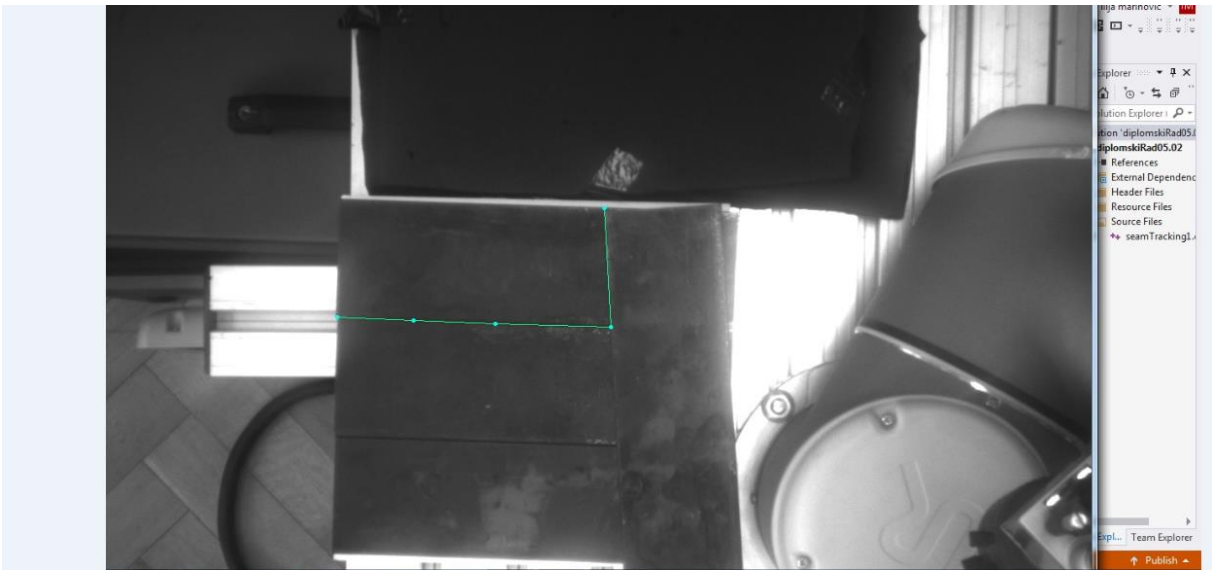
4.1.1 Analiza slike s gornje kamere

Kao što je već prethodno spomenuto, inicijalnu putanju robota određuje korisnik, a s obzirom da je to jedina zadaća ove kamere njezina konfiguracija je jednostavna. Sve što je potrebno je dobro fokusirati leću tako da su konture i predmet zavara vidljivi na slici nakon čega korisnik mišem može označiti željenu konturu. Također, prije svega je potrebno napraviti kalibraciju kamere na način kako je to navedeno u [9]. Visina na kojoj se kamera nalazi je uvijek ista tako da promjenom visine predmeta snimanja mijenja se fokus i kalibracijski omjer s gornje kamere, ali za potrebe ovog rada to je zanemareno, jer se pokaza da ne odstupa previše od stvarnih vrijednosti.

Točke koje korisnik označi na slici su pohranjene u vektor točaka te se iznos njihovih piksela množi s kalibracijskim omjerom kako bi se dobila pozicija te točke u koordinatnom sustavu robota. Program reagira tako da poziva funkciju *CallBackFunc()* te reagira na lijevi klik miša.

```
void CallBackFunc(int event, int x, int y, int flags, void* userdata)
{
      if (event == EVENT_LBUTTONDOWN)
      {
            Dio programa gdje se spremaju i iscrtavaju točke
       }
}
```

Nakon odabira točaka program će ih povezati i prikazati te upitati prikazuje li označena putanja uistinu konturu koju se želi pratiti.

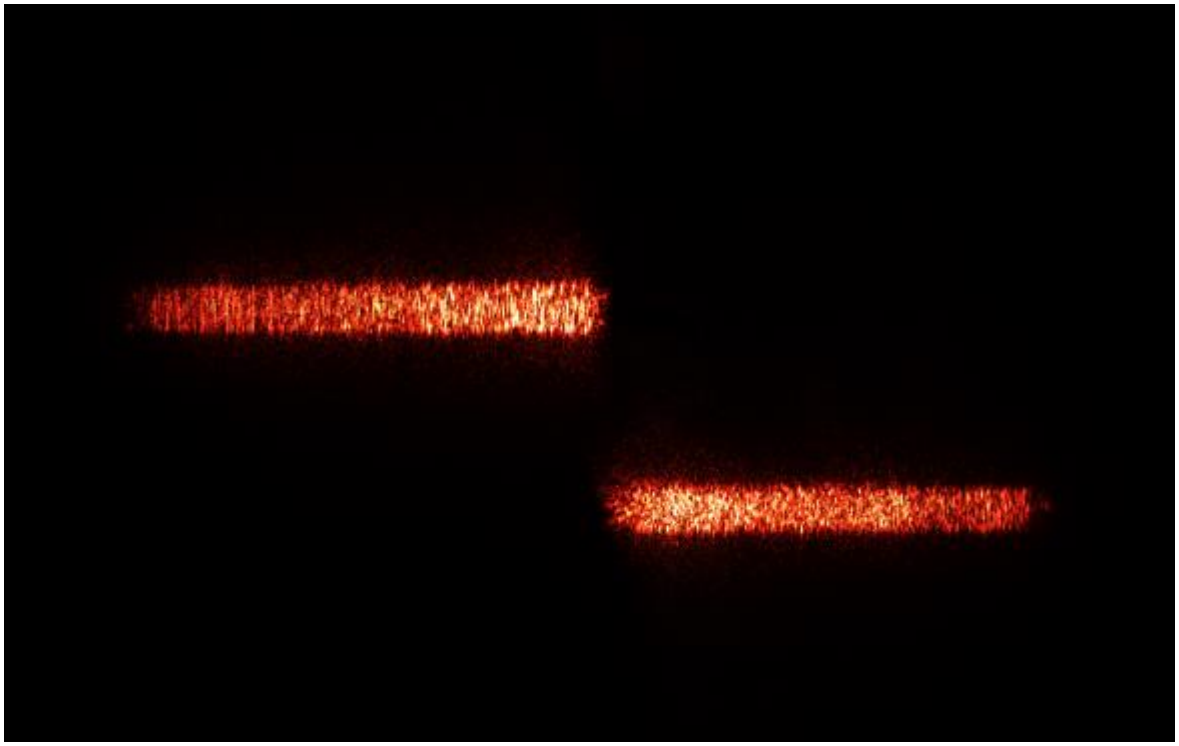


Slika 13: Prikaz označene putanje

Ukoliko je korisnik zadovoljan odabirom konture, program će izračunati kutove rotacije koje robot treba napraviti da bi ispratio konturu do kraja. Međutim, tu treba biti oprezan jer je potrebno izračunati kut s ispravnim predznakom rotacije u odnosu na koordinatni sustav robota.

4.1.2 Analiza slike s kamere koja snima zavar

Nakon što je završeno snimanje s gornjom kamerom program svoj fokus prebacuje na kameru koja snima konturu zavara.



Slika 14: Neprocesuirana slika s kamere

Slika 14 prikazuje neprocesuiranu sliku dohvaćenu kamerom kako bi se provela zamišljena analiza prvi korak je pretvoriti tu sliku u *grayscale* openCv knjižnica posjeduje funkciju ***cvtColor()*** koja čini upravo to prema sljedećoj formulaciji.

RGB[A] to Gray:Y←0.299*R+0.587*G+0.114*B

R,G i B predstavljaju redom crveni, zeleni i plavi kanal slike. Puna definicija funkcije glasi :

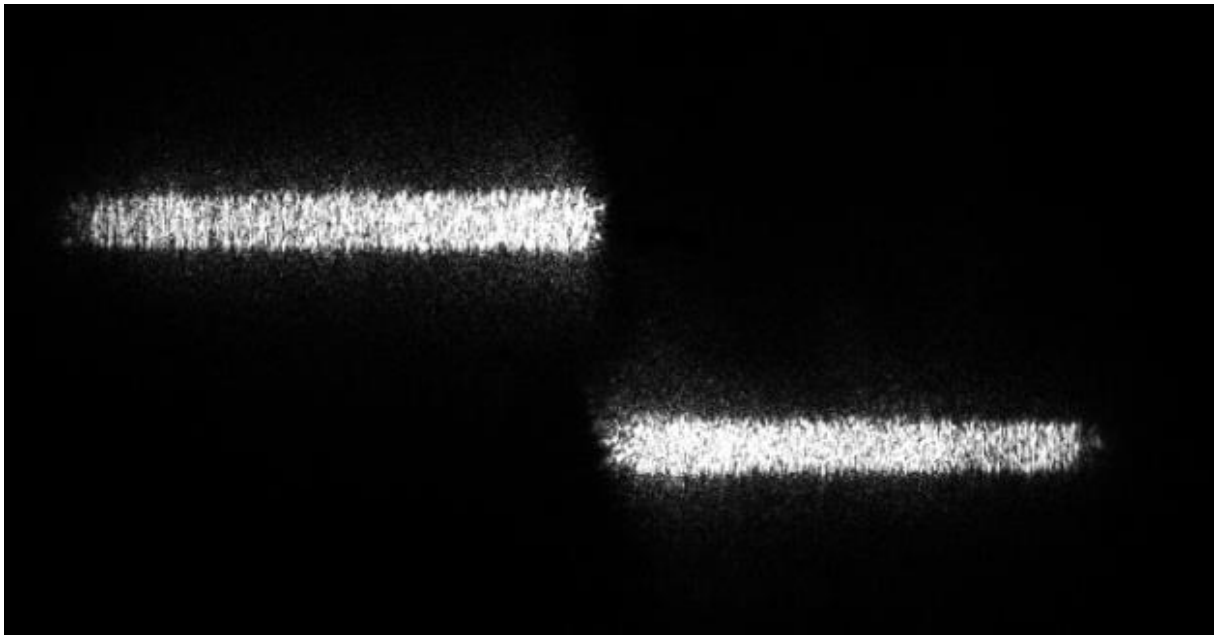
```
void cvtColor( InputArray src, OutputArray dst, int code, int dstCn = 0 )
```

Src tako čini ulaznu matica, odnosno slika, a *dst* izlazna slika dok je *int code* željena operacija pretvorbe. U tablici 4 je moguće vidjeti više vrsta pretvorbi koje su moguće ovom funkcijom.

Može se zamijetiti kako je u openCv-u zapis kanala slike obrnuti tako da je prvi kanal plavi (B), nakon čega idu zeleni(G) i crveni (R).

Tablica 4: Neke od vrsti pretvorbi [6]

Conversion code	Meaning
CV_BGR2RGB CV_RGB2BGR CV_RGBA2BGRA CV_BGRA2RGBA	Convert between RGB and BGR color spaces (with or without alpha channel)
CV_RGB2RGBA CV_BGR2BGRA	Add alpha channel to RGB or BGR image
CV_RGBA2RGB CV_BGRA2BGR	Remove alpha channel from RGB or BGR image
CV_RGB2BGRA CV_RGBA2BGR CV_BGRA2RGB CV_BGR2RGBA	Convert RGB to BGR color spaces while adding or removing alpha channel
CV_RGB2GRAY CV_BGR2GRAY	Convert RGB or BGR color spaces to grayscale
CV_GRAY2RGB CV_GRAY2BGR CV_RGBA2GRAY CV_BGRA2GRAY	Convert grayscale to RGB or BGR color spaces (optionally removing alpha channel in the process)
CV_GRAY2RGBA CV_GRAY2BGRA	Convert grayscale to RGB or BGR color spaces and add alpha channel
CV_RGB2BGR565 CV_BGR2BGR565 CV_BGR5652RGB CV_BGR5652BGR CV_RGBA2BGR565 CV_BGRA2BGR565 CV_BGR5652RGBA CV_BGR5652BGRA	Convert from RGB or BGR color space to BGR565 color representation with optional addition or removal of alpha channel (16-bit images)
CV_GRAY2BGR565 CV_BGR5652GRAY	Convert grayscale to BGR565 color representation or vice versa (16-bit images)

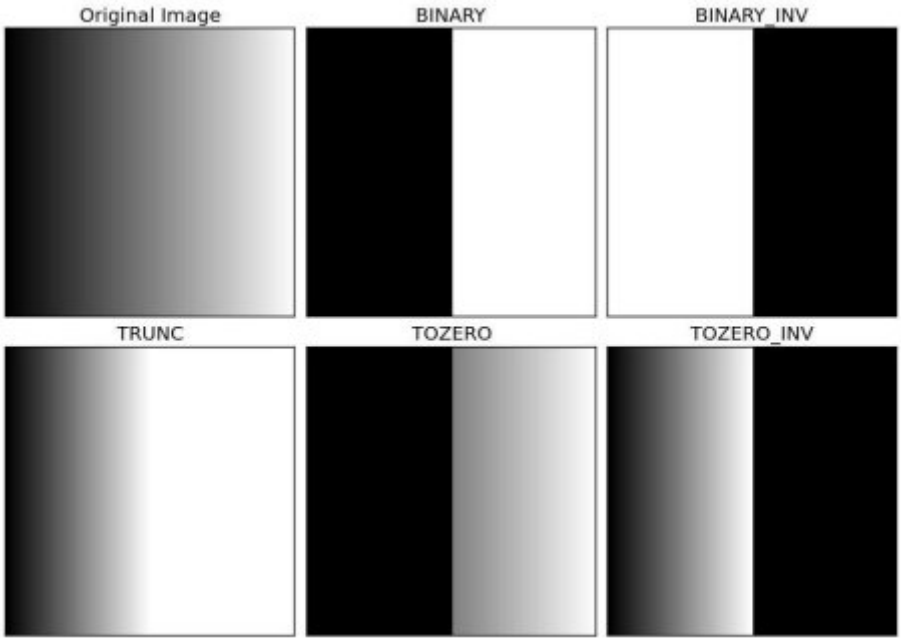


Slika 15: Grayscale slika

Slika 15 prikazuje sliku dobivenu nakon pretvorbe. Matematički gledano, slika dobivena nakon pretvorbe je matrica koja ima vrijednosti piksela od 1 do 255. Nije teško zamijetiti kako ova slika ima puno šuma kojega treba ukloniti. To se najlakše ostvaruje funkcijom prag *(eng. threshold).* Da bi se opće proveo thresholding potrebno je imati grayscale sliku no taj uvjet ovdje je već ispunjen. Threshold radi na slijedećem principu: postavlja se neka granična vrijednost thresholda (okidača) tako da se sve vrijednosti manje od njega pretvore u neku vrijednost npr. 0 što je crna boja, dok su pikseli veće vrijednosti postavljeni u bijelu odnosno vrijednosnu 1.

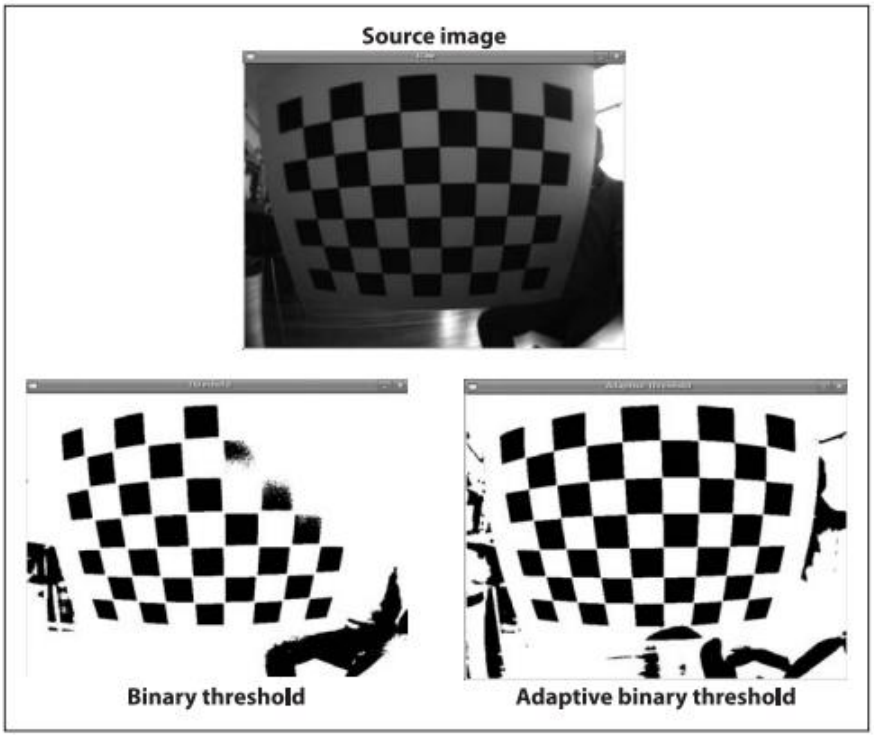
OpenCv knjižnica nudi više vrsta thresholdinga, a neka od njih su, slika 13. :

- CV_THRESH_BINARY
- CV_THRESH_BINARY_INV
- CV_THRESH_TRUNC
- CV_THRESH_TOZERO
- CV_THRESH_TOZERO_INV



Slika 16: Usporedba različitih vrsta thresholda[4]

Uz to postoji još i metoda adaptivnog thresholda, a to podrazumijeva da se threshold vrijednost mijenja ovisno o regiji na slici. Najbolje objašnjenje za takvo što bi bilo kad bi se radilo na procesuiranju slike kojoj zbog različitog osvjetljenja svi dijelovi slike nisu jednako istaknuti, a cilj se izvući neku značajku. Takav primjer je prikazan u slici 14.

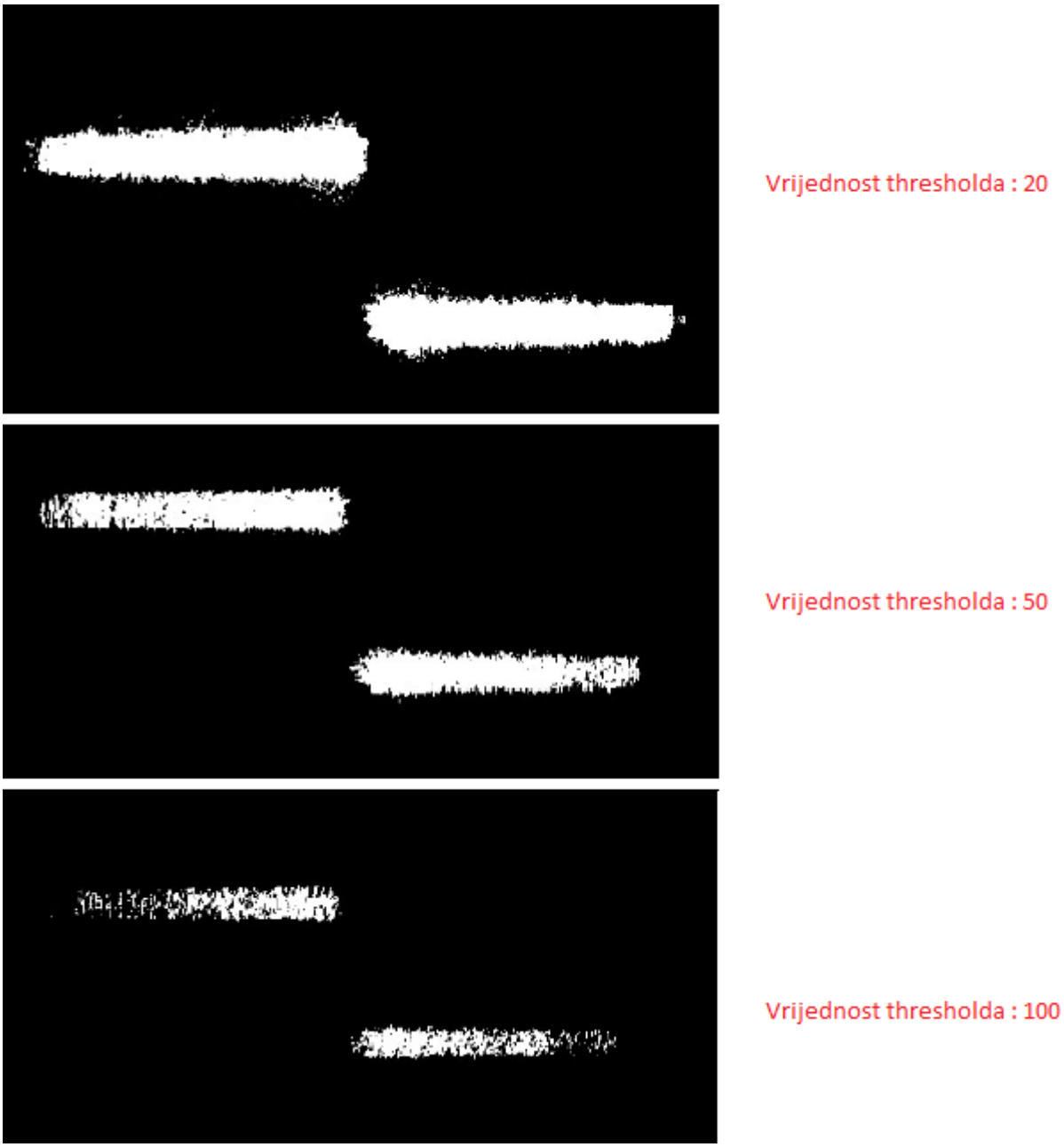


Slika 17:Usporedba klasičnog i adaptivnog thresholda na slici s nejednakim osvjetljenjem

Pogledom na sliku 15 primjećuje se da je osvjetljenje ujednačeno te da nema potreba za adaptivnim thresholdom stoga se provodi obični binarni threshold.

```
double threshold( InputArray src, OutputArray dst, double thresh, double
maxval, int type );
```

Src i *dst* opet čine ulazne odnosno izlazne slike, a *thresh* vrijednost praga. *Maxval* je maksimalna vrijednost piksela, a *type* vrsta *thresholda* je THRESH_BINARY.



Slika 18: Različite vrijednosti pragova na slici

Iz gore provedenih testiranja nije teško zaključiti kako je za ovu primjenu bolje uzeti nižu vrijednost praga, ali opet treba biti oprezan da se ne dobije slika s previše nebitnih značajki. Nakon što je slika „očišćena“ od šuma može se prijeći na sljedeći korak, a to je traženje kontura. Kontura se može definirati kao skup točaka koji opisuje neku krivulju [6].

```
void findContours(InputOutputArray slika, OutputArrayOfArrays contours, OutputArray hierarchy, int mode, int method, Point offset=Point())
```

Parametri funkcije:

Slika – je ulazna slika u funkciju, trebala bi biti 8-bitna jednokanalna slika koja je već predprocesirana na neki način, može se koristiti neka od sljedećih metoda compare() , inRange(), threshold(), adaptiveThreshold(), Canny() i druge. Vrijednosti piksela koji su 0 ostaju nula dok se drugi postavljaju u 1.

Contours – snimljena kontura, svaka kontura se sprema u vektor točaka.

Hierarchy – je opcionalni element, a govori o topologiji slike odnosno o prioritetima kontura.

Mode – način na koji će se izuzeti konture:

- CV_RETR_EXTERNAL - prikuplja podatke samo od vanjskih točaka.
- CV_RETR_LIST – prikuplja sve podatke bez obzira na topološki spektar.
- CV_RETR_TREE- prikuplja sve podatke uključujući i njihov topološki spektar.

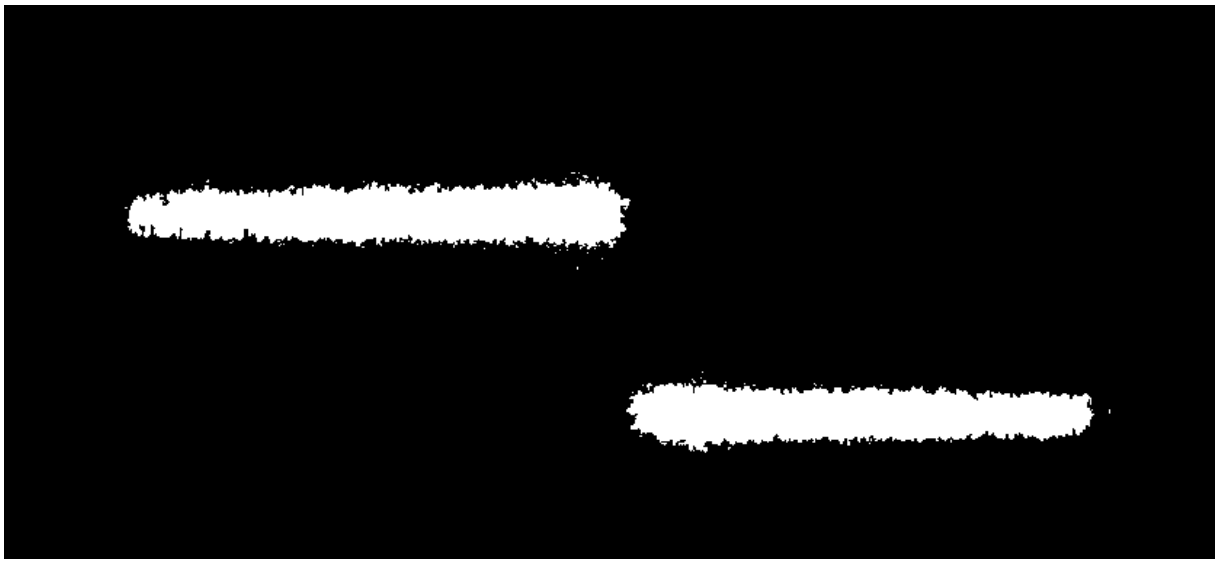
Metode :

- CV_CHAIN_APPROX_NONE – pohranjuje sve točke konture.
- CV_CHAIN_APPROX_SIMPLE – kompresira podatke po horizontalnoj i vertikalnoj ravnini i ostavlja samo krajnje točke konture.

Nakon što je kontura određena, potrebno ju je označiti i ispuniti. Funkcija *drawContours()* može iscrtati vanjski obrub konture ili ju ispuniti. Za ovu primjenu potrebno je ispuniti konturu.

void drawContours(InputOutputArray ***slika,*** *InputArrayOfArrays* ***contours,*** *int* ***contourIdx,*** *const* ***Scalar&*** ***color,*** *int* ***thickness****=1, int lineType=8, InputArray* ***hierarchy****=noArray(),int* ***maxLevel****=INT_MAX, Point offset=Point())*

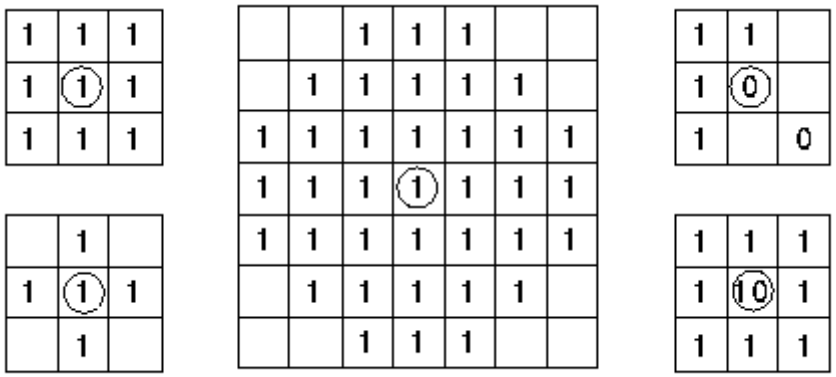
Parametar *slika* definira sliku koju procesuiramo, *contours* je vektor točaka prethodno spremljene konture, a *contourIdx* je parametar koji određuje koje će konture biti zahvaćene analizom te ukoliko je taj parametar negativan sve konture su obuhvaćene. *Thickness* određuje debljinu linije, a ako je vrijednost negativna ispunjava se unutrašnjost konture. Slika 16 prikazuje konturu nakon ispuna.



Slika 19: Ispunjena kontura

Međutim, i dalje se može primijetiti kako su prisutni izolirani bijeli pikseli te postoji mnogo „oštrih“ vrhova duž konture, stoga je neophodno provesti neku od morfoloških operacija u cilju uklanjanja ovih smetnji. Morfološka transformacija na slici je jednostavna operacija koja ima dva elementa, prvi, ulaznu sliku za koja je u pravilu binarna i drugi, strukturalni element ili jednostavnije kernel. Kernel je matrica koja „korigira“ ulaznu u sliku ovisno o svom izgledu odnosno operaciji koju provodi. Naziv kernel se u terminu vizijskih sustava uglavnom koristi kod konvolucije tako da će se u ovom radu ubuduće koristiti naziv strukturni element. Neke od vrsta morfoloških operaciju su :

- Dilation – širi konturu
- Erosion – stanjuje konturu
- Opening – uklanja piksele s granice konture
- Skeleton – ostavlja samo tanki kostur konture



Slika 20: Neki od primjera strukturalnog elementa [7]

S obzirom da se na slici 19 vidi kako su nepravilnosti vezane za rub konture i izolirane bijele piksele, koristiti će se morfološka operacija otvaranja konture koja će za rezultat dati zaglađenu konturu. Prvi korak je napraviti strukturalni element :

```
Mat element = getStructuringElement(MORPH_ELLIPSE, Size(7, 7));
```

Ovaj strukturalni element je matrica 7x7 koja ima jedinice u elipsoidnom redoslijedu.

Nakon što se napravi strukturalni element, moguće je provesti morfološku operaciju slijedećom funkcijom:

```
morphologyEx(Ulazna_slika, Izlazna_slika, CV_MOP_OPEN, element);
```

Slika 21:Izlazna slika nakon analize

Usporedbom slika 14. i 21. dolazi vidi se iz vizijski beznačajne slike došlo do slike koja se može koristiti za daljnju analizu. Ovaj korak je bio potreban jer sad postoji uređena slika s dvije konture koje vjerno prikazuju konturu preklopnog zavara. U idućem poglavlju bit će prikazano kako su izvučene pojedine značajke kao što su visina zavara i duljina kontura koja direktno utječe na korekciju robota kod praćenja konture.

4.1.3 Značajke slike zavara

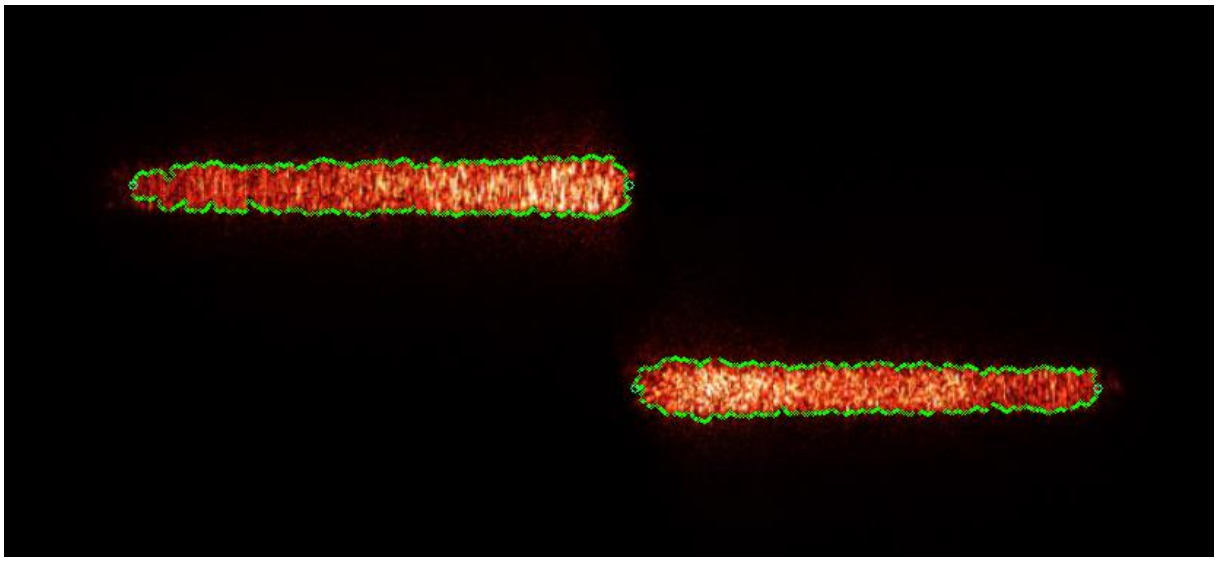
Kako bi se provela kalibracija kamere te tako dobili podaci sa svim značajkama potrebno je dodatno analizirati sliku. Želja je kroz obje vidljive konture na slici provući liniju kroz njihova središta. To je ostvareno tako što će se metodom najmanjih kvadrata kroz sve točke konture provući pravac.

Funkcija koja to omogućava je ***fitLine()*** puna argumentacija funkcije glasi:

```
fitLine(Uzazni_set_točaka, Izlaz_funkcije, int distType, double param, double reps, double aeps)
```

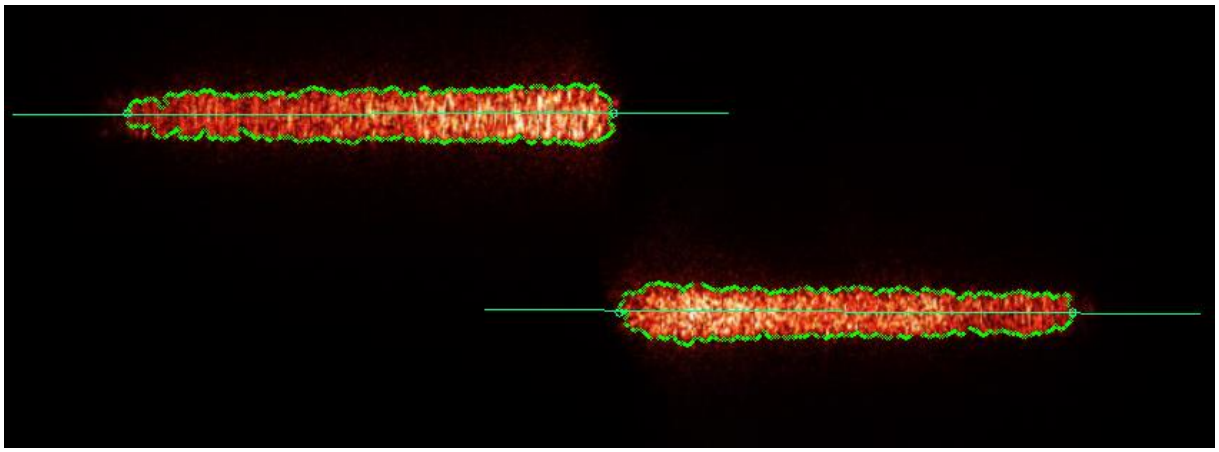
- Ulazni_set_točaka – su sve točke na pripadajućoj konturi
- Izlaz_fukncije - slučaju 2D slike to je vektor od 4 elementa (kao što je Vec4f) - (vx, vy, x0, y0), gdje je (vx, vy) normalizirani vektor kolinearan liniji, a (x0, y0) je točka na liniji.
- distType – metoda koja se koristi, za metodu najmanjih kvadrata parametar je distType=CV_DIST_L2, za više pogledati literaturu [8].
- ostali parametri se odnose na preciznost i točnost fukncije

Slika 22 prikazuje sve točke koje ulaze u funkciju.



Slika 22: Točke kroz koje se provlači pravac

Također je potrebno označiti i početnu i krajnju točku konture koja se nalazi na centralnoj liniji, nakon toga jednostavnom funkcijom *cv::line()* se nacrta tražena linija.



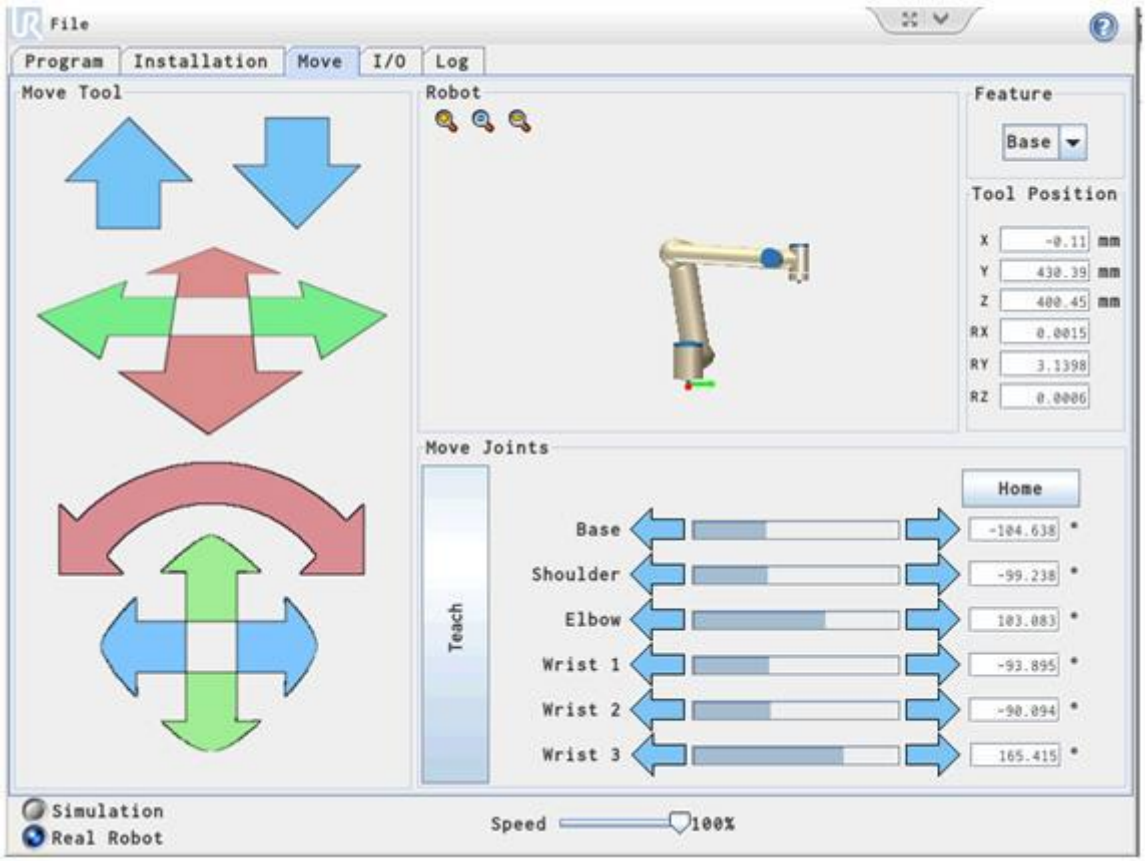
Slika 23: Prikaz linija kroz konture

Ova analiza omogućuje da se dobiju podatci o visini zavara ili horizontalnoj korekciji robota. Međutim svi ti podatci su u pikselima i oni kao takvi su potpuno neupotrebljivi za robota. Kako bi se ti podatci pravilno iskoristili potrebno je provesti kalibraciju kamere. Program za kalibraciju se neće prolaziti detaljno jer je analiza slike i značajki ista, no razvijen je etalon za kalibraciju koji je prikazan na slici 24. S obzirom da će se snimanje uvijek odvijati s iste visine u odnosu na predmet snimanja program za kalibraciju funkcionira poprilično jednostavno. Poznate dimenzije „stepenica“ dijelimo s brojem piksela koje one zauzimaju na slici te dobivamo kalibracijski omjer s kojim množimo broj piksela na slici za buduće slučajeve. Kako bi kalibracija bila što točnija napravljeno je više stepenica tako da se u obzir uzimaju prosječne vrijednosti. Ovakvim pristupom dobila se greška manja od 0.5 mm što je prihvatljivo za ovaj rad.

Slika 24: Kalibracijski blok

4.2 PROGRAMIRANJE ROBOTA

Programirati robota je moguće na više načina. UR robot dolazi s privjeskom za učenje, na kojem je moguće programirati robota korištenjem standardnih naredbi u polyscope grafičkom okruženju slika 25.

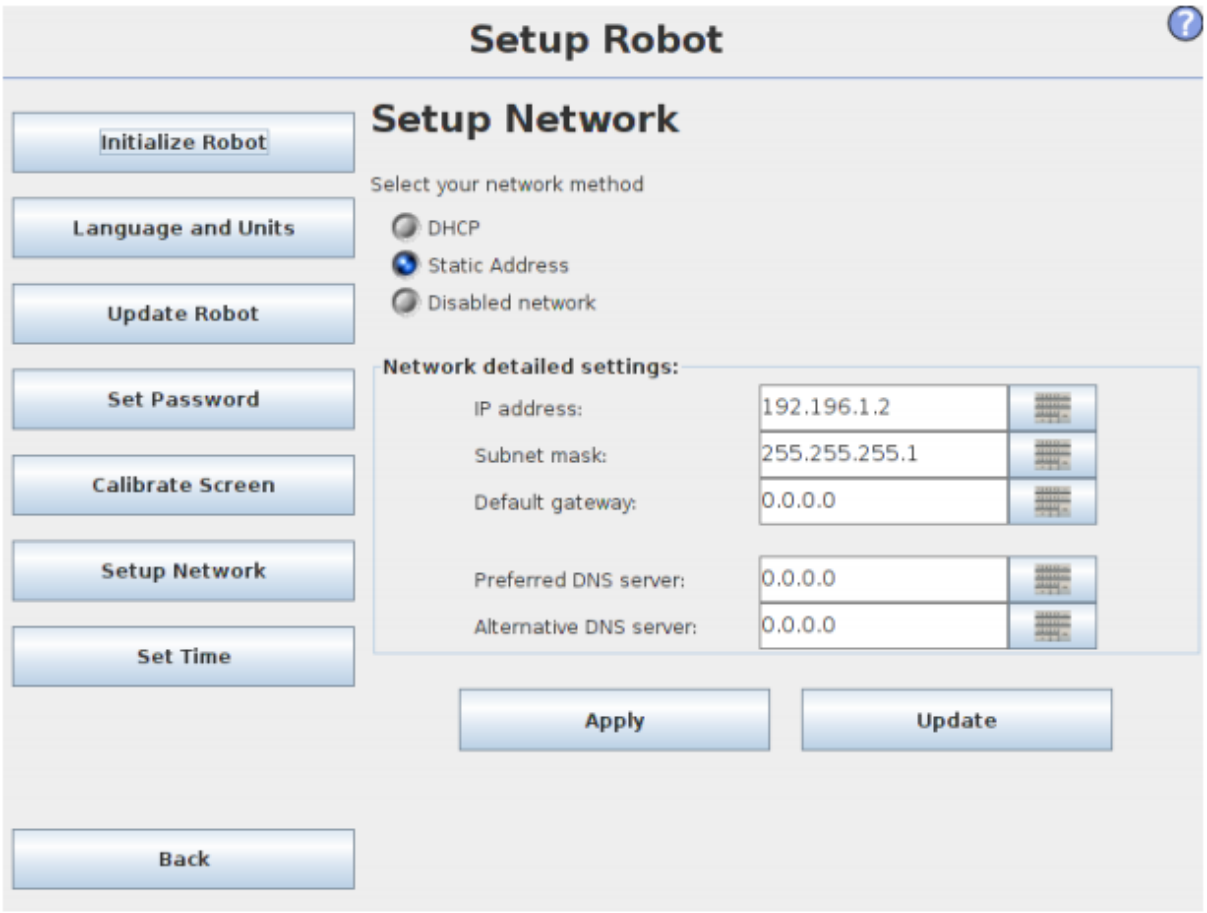


Slika 25: Grafičko sučelje na privjesku za učenje

Uz to moguće je pokrenuti isti program na računalu te pripremiti program na računalu koji se kasnije prebaci na robota i učita. Uz to postoji još i programiranje putem skripte koja se može napisati u nekom od programskih jezika npr. Matlab ili C++ te putem mrežne komunikacije slati robotu direktno na izvršavanje za što je potrebno imati konfiguriranu komunikaciju klijent – poslužitelj, no o tome će biti riječi u idućem poglavlju.

4.2.1 Komunikacija robot - računalo

Već je ranije spomenuto da su korišteni prilagođeni programi za komunikaciju razvijeni u literaturi [10]. Prvi korak u ostvarivanju komunikacije je postaviti IP adresu robota, a to je moguće napraviti na privjesku za učenje odabirom izbornika „setup network“ te postaviti IP adresu ranije dodijeljenu robotu 192.168.0.60 , također treba obratiti pozornost da sve adrese pripadaju istoj mreži *eng. Subnet.*



Slika 26: Postavljanje IP adrese na robotu

Kako bi komunikacija robota bila moguća u oba smjera (da je robot server, a računalo klijent i obratno) robot raspolaže s 3 karakteristična servera na portovima 30001, 30002, 30003., a redom se nazivaju primarni, sekundarni i real time server.

Tablica 5:Serveri na UR kontroleru [11]

	Primarni server	Sekundarni server	Real –Time server
Port	30001	30002	30003
Frekvencija [Hz]	10	10	500
Primanje	URScript naredbe	URScript naredbe	URScript naredbe
Slanje	Stanje robota i dodatne poruke	Stanje robota	Stanje robota

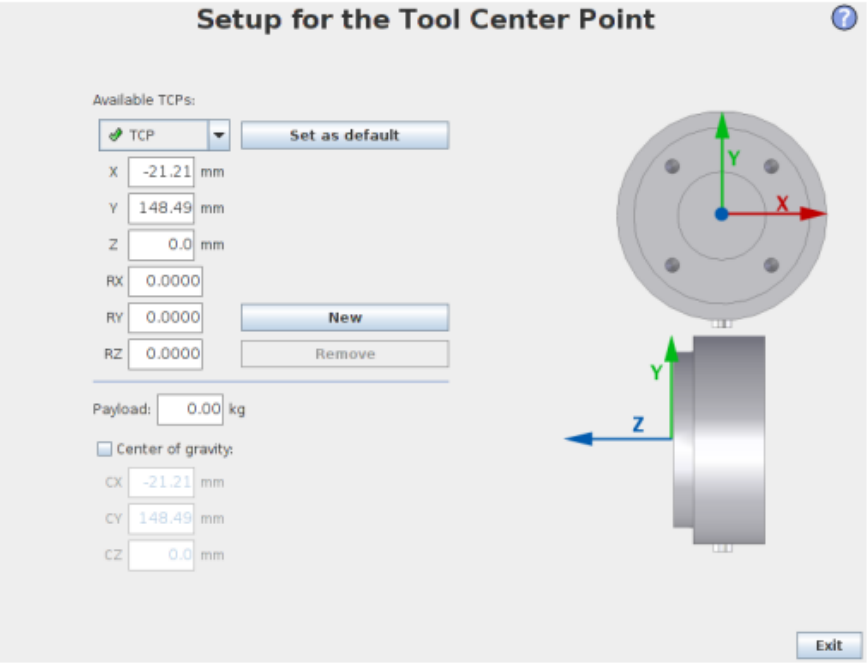
U ovom radu korišteni su serveri na portovima 30002 za slučaj kada računalo šalje robotu podatke i 30003 u slučaju kada računalo očekuje podatke o trenutnom stanju robota. Robot neprestano šalje skup različitih podataka kao što su pozicije zglobova, temperature, sile i ostalo međutim u ovom radu jedina potrebna stavka od robota je njegov trenutni položaj vrha alata u axis – angle zapisu.

4.1.2 Kalibracija vrha alata

Kako bi kretnje robota bilo što preciznije, potrebno je odrediti dobru poziciju vrha alata, pozicija vrha alata napravljena je po modelu razvijenom u [10]. To znači da se vrh alata robota treba dovesti do nekog šiljastog etalona te pristupiti njegovom vrhu iz četiri različite točke. S obzirom da je vrh alata u ovom radu zapravo točka u prostoru koja je udaljena od mehaničkih komponenti napravljen je improvizirani vrh alata koji se postavljen u produžetku laserske zrake okomito na predmet snimanja. Program funkcionira na sljedeći način:

1. Vrh alata robota se približi vrhu etalona iz jednog položaja
2. Robot očita trenutnu poziciju zglobova
3. Robot zahtjeva pomicanje iz drugog položaja

Ovaj postupak se ponavlja za 4 točke nakon čega se prema [12] računa pozicija vrha alata koja je predstavljena kao vrijednost X,Y i Z odstupanja točke vrha alata od vrha zadnjeg zgloba na robotu. Postavljanje vrha alata moguće je napraviti direktno na privjesku za učenje kao što je to prikazano na slici 28.



Slika 27: Postavljanje TCP-a na privjesku za učenje

U skriptnom zapisu to izgleda ovako: `set_tcp(p[0.05795,-0.09082,0.2893,0,0,0])`

4.2.3 Skriptno programiranje

Kako bi robot ispravno primao informacije od računala, potrebno je poštovati neka pravila programiranja:

1. Podaci o poziciji i orijentaciji moraju biti zapisani u zagradi.
2. Svaki podatak mora biti razdvojen zarezom.
3. Na kraju izraza koji se šalje potrebno je dodati oznaku '\n' kako bi se svako slanje izvršilo u novom redu.

Uslijed stalnog slanja robot neće izvršavati novodolazeće naredbe o poziciji sve dok prvotno dobivenu naredbu ne izvrši do kraja, međutim preporuka je da server pošalje novu naredbu tek nakon što dobije povratni signal od klijenta da je izvršena prethodna operacija kako bi se povećala stabilnost sustava. Uz to potrebno je znati kako se deklariraju varijable i funkcije u skriptnom zapisu, tako da će se u slijedećem tekstu proći kroz neke od funkcija korištenih u ovom radu.

Funkcija u skriptnom zapisu izgleda ovako:

```
def add(a,b):     // gdje je add naziv funkcije, a (a,b) parametri
                  // ako je nema parametara ostavlja se prazna zagrada ()
return a+b        // sadržaj funkcije
end               // svaka funkcija treba završiti naredbom end
```

Svaku varijablu unutar programa je potrebno deklarirati ili kao globalnu ili kao lokalnu, a razlika je u tome što kod lokalnog (`local`) deklariranja varijable kontroler tu varijablu uistinu gleda kao lokalnu i nema problema ukoliko u funkciji postoji i globalna (`global`) varijabla istog naziva. Stoga ,ukoliko se želi deklarirati globalna varijabla npr. neke pozicije to izgleda ovako:

```
global naziv_varijable = p[X,Y,Z,Rx,Ry,Rz]
```

Deklariranje gibanja u skriptnom zapisu je analogno zapisu na upravljačkom privjesku uz dodatak brzina i ubrzanja. Pa tako se naredba za linearno gibanje u odnosu na vrh alata zapisuje na slijedeći način:

```
movel(pose, a=1.2, v=0.25, t=0, r=0)
```

Parametri:

`pose:` pozicija u koju robot treba doći može se zapisati kao varijabla ili direktno u axis-angle zapisu.

`a:` ubrzanje robota

`v:` brzina kretanja vrha alata

`t:` vrijeme gibanja

`r:` radijus interpolacija

Analogno linearnom zapisu izvode se i zapisi za kružno gibanje i slobodno zglobno gibanje.

Kako bi se u svakom trenutku znala pozicija vrha alata postoji funkcija *get_actual_tcp_pose()*, dok je njen izlaz vektorski zapis položaja vrha alata u axis-angle zapisu.

4.2.4 Transformacija položaja

Kako bi se ostvarilo gibanje u odnosu na neku prethodnu poziciju, potrebno je provesti transformaciju položaja. Postoji više vrsta transformacija položaja, a ovdje će biti spomenute samo neke. Pogledom u literaturu [13] može se vidjeti ostatak funkcija.

Naredba:

`resulting_pose = pose_trans(p_feature, p_wrt_feature)`

Parametri:

`p_feature` – početna pozicija.

`p_wrt_feature` – promjena pozicije relativno na početnu poziciju

`pose_trans()` – funkcija transformacije u kojoj prvi argument (`p_feature`) služi za transformaciju drugog argumenta `p_wrt_feature.`

Naredba:

`pose_add(p_1, p_2)` – je naredba koja zbraja pozicije na način da ih gleda kao dvije odvojene cjeline. P sadrži x , y i z vrijednosti te se one uistinu zbrajaju matematički dok R sadrži rotacijske vrijednosti Rx, Ry i Rz koje se matrično množe.

```
p_3.P = p_1.P + p_2.P
p_3.R = p_1.R * p_2.R
```

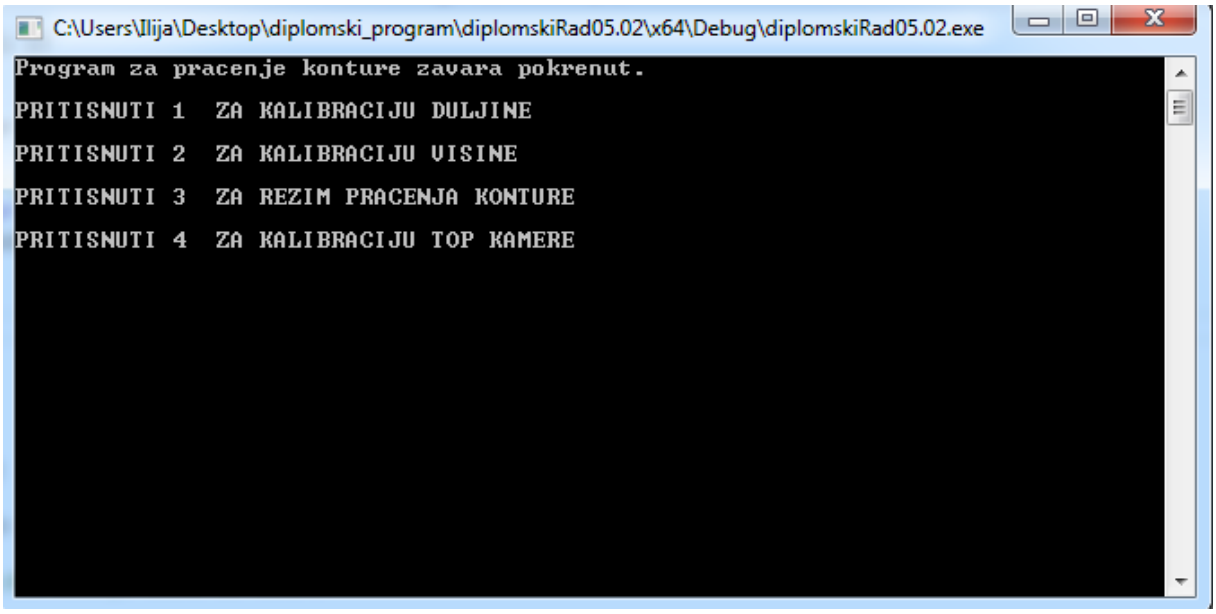
Ova naredba je korištena za računanje kretanja u x-y ravnini iznad predmeta snimanja. Sve odabrane točke posjeduju svoje koordinate na slici, a kalibracijom je dobivena korelacija između piksela na slici i stvarne udaljenosti tako da se svaka nova pozicija računa tako što se na ishodište dodaje vrijednost odmaka točke od ishodišta. Programerski je to izvedeno na slijedeći način:

```
global pose_wrt_feature=p["<<X_cord[1]<<","<< Y_cord[1] << ",0,0,0,0]\n";
global feature_wrt_base=p[0.1731,-0.328,0.03193,0.1006,2.3122,0.0007]\n";
global pose_wrt_base=pose_add(feature_wrt_base, pose_wrt_feature)\n";
```

Vektori točaka `X_cord[]` i `Y_cord[]` su vrijednosti pomaka koje se zbrajaju na ishodišnu poziciju robota (`feature_wrt_base`).

4.3 KORISNIČKO SUČELJE

Aplikacija je napravljena kao windows konzolna aplikacija i samim pokretanjem programa dolazi se do izbornika za odabir režima rada slika 28.



Slika 28: Izbornik

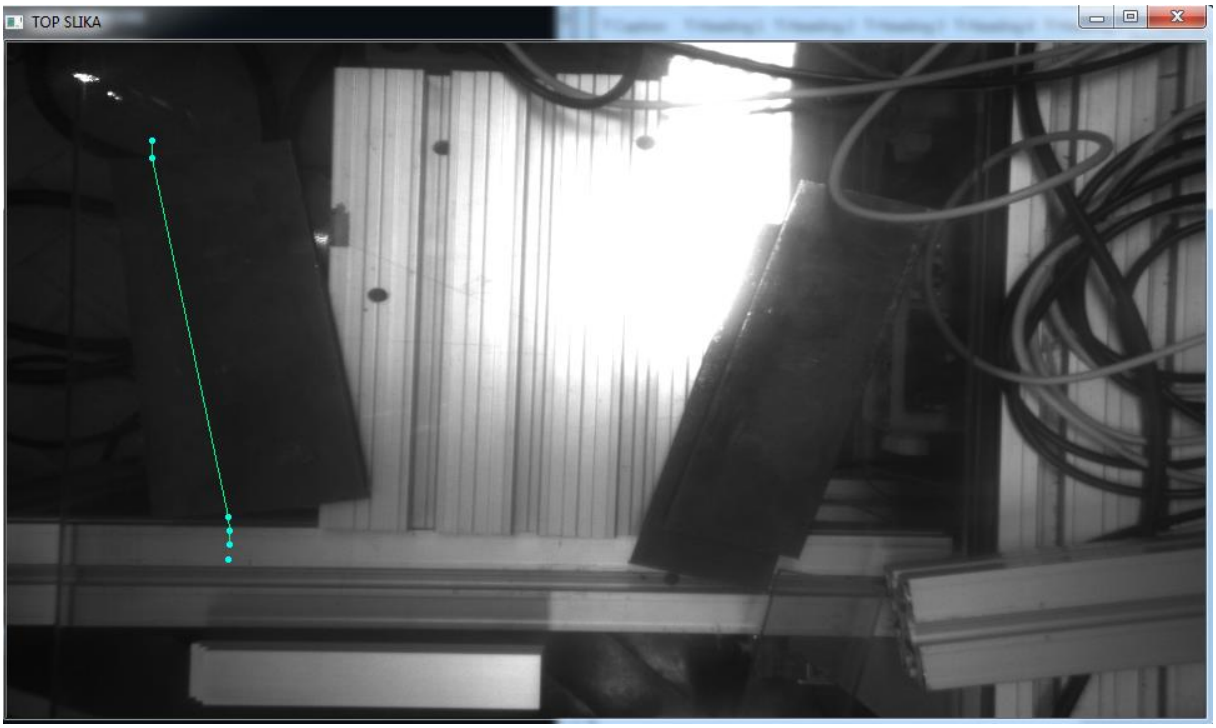
Odabirom:

1. Pokreće program koji služi za kalibraciju visine, a radi tako da se kao predmet snimanja postavi kalibracijski etalon iz slike 21 nakon čega se dohvaća slika s kamere koja gleda laserska zraka na mjestu zavara. Kalibracijski blok je visine 8mm, dubine 5mm i ima dvije stepenice. Program računa udaljenosti između tri razlomljene laserske zrake prikazane na slici 30 u pikselima nakon čega se računa prosječna vrijednost svake od tih udaljenosti po principu da se računa prosječna vrijednost na temelju jedne stepenice od 8mm, a potom toga prve dvije sa skokom od 16mm, nakon čega se računa prosječna vrijednost od ta dva omjera. Po završetku programa kreirana je tekstualna datoteka koja sadrži vrijednost kalibracijskog omjera te se on učitava pri pokretanju režima rada praćenja konture zavara. Ovom metodom kalibracije dobila se točnost računanja visine s pogreškom +/- 0.5 mm

Slika 29: Prikaz slike s kalibracijskog bloka

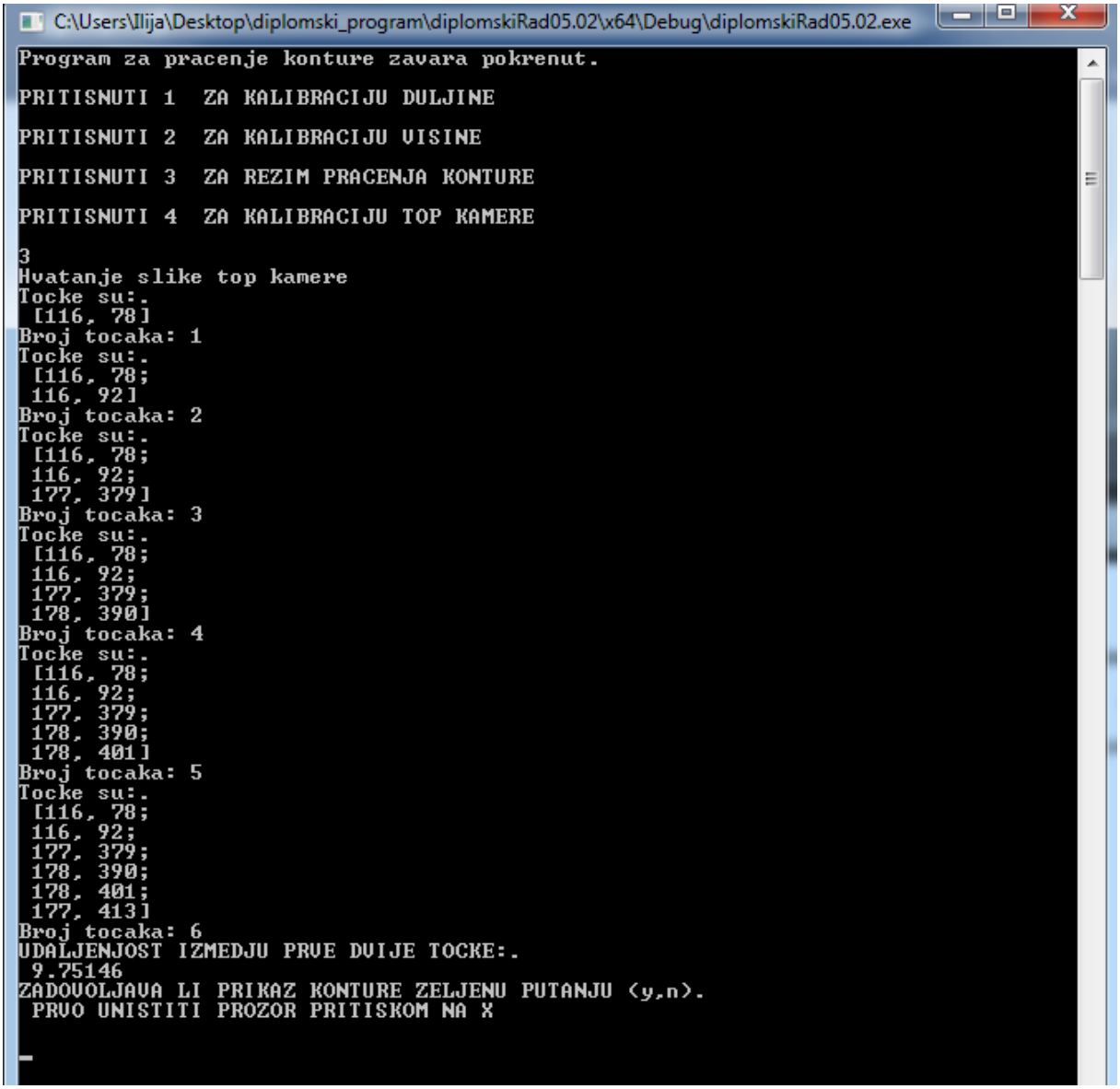
2. Gotovo analogno prethodnom slučaju kalibracije visine provodi se i kalibracija duljine. Također se postavlja kalibracijski etalon ispod lasera te se dohvaća slika s kamere koja gleda lasersku zraku. Program računa duljine svake linije u pikselima te jednako kao i u prethodno slučaju računa prosječne vrijednosti omjera za svaku liniju i njihove zbrojeve, nakon čega ponovno računa prosječnu vrijednost toga i zapisuje ju u tekstualnu datoteku iz koje program dohvaća podatke o kalibraciji u slučaju režima rada kod praćenja konture zavara. Ovom metodom kalibracije duljine dobila se točnost od +/- 0.5 mm što zadovoljava standarde u ovom radu.

3. Izborom ovog režima rada pokreće se program koji je i tema ovoga rada, a to je praćenje konture zavara. Prethodno je rečeno, a vidljivo je i iz algoritma predstavljenog u točki 5.1, kako se kod ovog režima rada prvo dohvaća slika s gornje kamere koja prikazuje radni prostor. Predmet snimanja se položi u radni prostor te se korisniku prikazuje slika s gornje kamere na kojoj se vidi predmet. Korisnik pritiscima lijeve tipke na prozoru slike označavaju više točaka konture nakon čega se linijama povežu označene točke slika 30.



Slika 30: Prikaz snimljene konture

Ukoliko je korisnik zadovoljan prikazanom konturom program računa pozicije točaka loma te kutove. Funkcije koje su računaju to su `racunanjeKuta()` i `scalingFaktor()`, a prikazane su u kodu kao prilog.

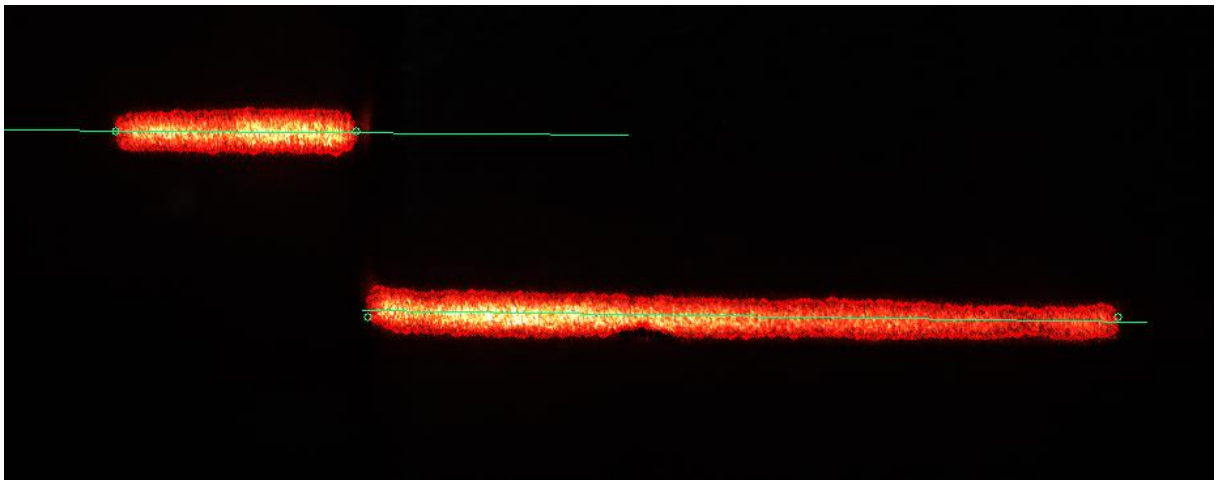


Slika 31: Korisničko sučelje nakon odabira konture

S druge strane, ukoliko korisnik nije zadovoljan označenom putanjom, unosom vrijednosti „n“ dolazi u mogućnost ponovnog označavanja željene konture. Konačnim odabirom putanje i računanjem svih potrebnih vrijednosti program kreće u svoje inicijalno snimanje po unesenoj putanji. Taj program se u programu naziva *slanjeTranslacija* i napravljen je kao samostalni potprogram koji se poziva u trenutku odabira željene konture. Izgled tog programa koji je napisan u skriptnom zapisu je slijedeći .

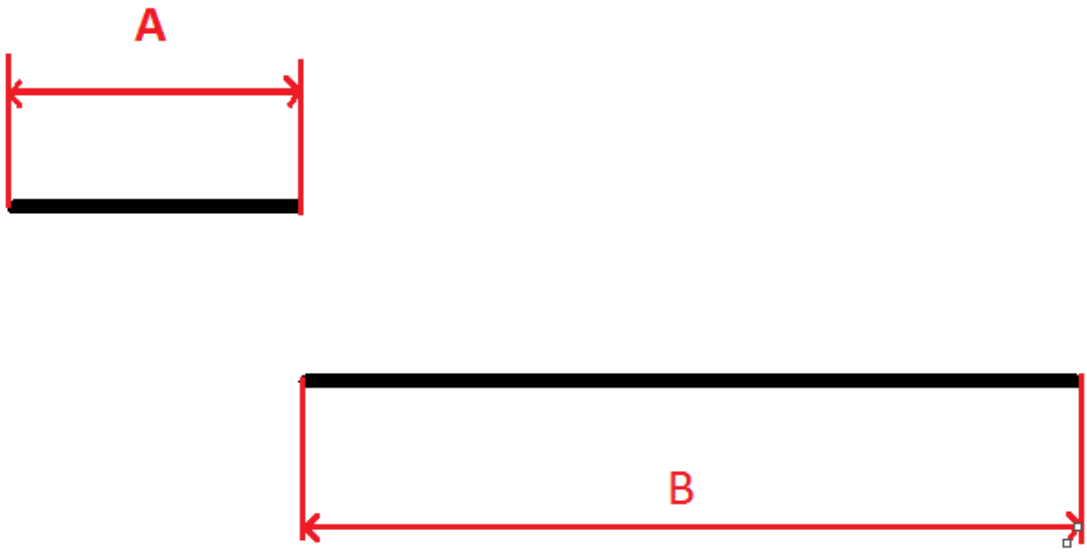
```
program.str("");
program << "def ilija():\n";
program << "set_tcp(p[0.05795,-0.09082,0.2893,0,0,0])\n";
program << "global pose_wrt_feature = p[" << X_cord[0] << "," << Y_cord[0] <<
",0,0,0,0]\n";
program << "global feature_wrt_base = p[0.1731, -0.328, 0.03193,-
0.1006,2.3122,0.0007]\n";
program << "global pose_wrt_base = pose_add(feature_wrt_base, pose_wrt_feature)\n";
program << "movel(pose_wrt_base,a=0.01,v=0.01)\n";

program << "global pose_wrt_feature = p[" << X_cord[1] << "," << Y_cord[1] <<
",0,0,0,0]\n";
program << "global feature_wrt_base = p[0.1731, -0.328,0.03193,-
0.1006,2.3122,0.0007]\n";
program << "global pose_wrt_base = pose_add(feature_wrt_base, pose_wrt_feature)\n";
program << "movel(pose_wrt_base,a=0.01,v=0.01)\n";
program << "movel(p[0.1731+" << X_cord[1] << ", - 0.328 +" << Y_cord[1] << ",0.03193,"
<< Rx_TCP_1 << "," << Ry_TCP_1 << "," << Rz_TCP_1 << "],0.01,0.01)\n";

program << "global pose_wrt_feature = p[" << X_cord[2] << "," << Y_cord[2] <<
",0,0,0,0]\n";
program << "global feature_wrt_base = p[0.1731, -0.328, 0.03193," << Rx_TCP_1 << ","
<< Ry_TCP_1 << "," << Rz_TCP_1 << "]\n";
program << "global pose_wrt_base = pose_add(feature_wrt_base, pose_wrt_feature)\n";
program << "movel(pose_wrt_base,a=0.01,v=0.01)\n";
program << "movel(p[0.1731+" << X_cord[2] << ", - 0.328 +" << Y_cord[2] << ",0.03193,"
<< Rx_TCP_1 << "," << Ry_TCP_1 << "," << Rz_TCP_1 << "],0.01,0.01)\n";

program << "global pose_wrt_feature = p[" << X_cord[3] << "," << Y_cord[3] <<
",0,0,0,0]\n";
program << "global feature_wrt_base = p[0.1731, -0.328, 0.03193," << Rx_TCP_1 << ","
<< Ry_TCP_1 << "," << Rz_TCP_1 << "]\n";
program << "global pose_wrt_base = pose_add(feature_wrt_base, pose_wrt_feature)\n";
program << "movel(pose_wrt_base,a=0.01,v=0.01)\n";
program << "movel(p[0.1731+" << X_cord[3] << ", - 0.328 +" << Y_cord[3] << ",0.03193,"
<< Rx_TCP_1 << "," << Ry_TCP_1 << "," << Rz_TCP_1 << "],0.01,0.01)\n";

program << "global pose_wrt_feature = p[" << X_cord[4] << "," << Y_cord[4] <<
",0,0,0,0]\n";
program << "global feature_wrt_base = p[0.1731, -0.328, 0.03193," << Rx_TCP_1 << ","
<< Ry_TCP_1 << "," << Rz_TCP_1 << "]\n";
program << "global pose_wrt_base = pose_add(feature_wrt_base, pose_wrt_feature)\n";
program << "movel(pose_wrt_base,a=0.01,v=0.01)\n";
program << "movel(p[0.1731+" << X_cord[4] << ", - 0.328 +" << Y_cord[4] << ",0.03193,"
<< Rx_TCP_1 << "," << Ry_TCP_1 << "," << Rz_TCP_1 << "],0.01,0.01)\n";

program << "global pose_wrt_feature = p[" << X_cord[5] << "," << Y_cord[5] <<
",0,0,0,0]\n";
program << "global feature_wrt_base = p[0.1731, -0.328, 0.03193," << Rx_TCP_1 << ","
<< Ry_TCP_1 << "," << Rz_TCP_1 << "]\n";
program << "global pose_wrt_base = pose_add(feature_wrt_base, pose_wrt_feature)\n";
program << "movel(pose_wrt_base,a=0.01,v=0.01)\n";
program << "end\n";
program << "end\n";
```

Simultano s pokretanjem programa za kretnju robota po prethodno definiranoj putanji pokreće se i program za snimanje laserske zrake koja pada na konturu zavara. Program je koncipiran tako da neće reagirati dok god na slici ne pojavi prikaz zavara kao dvije odvojene konture kao što je to bio slučaj u prethodnim poglavljima. U trenutku kada robot s kamerom dođe iznad konture zavara na zaslonu prikazuje prozor prikazan na slici 32.



Slika 32: Prikaz laserske zrake u režimu snimanja konture

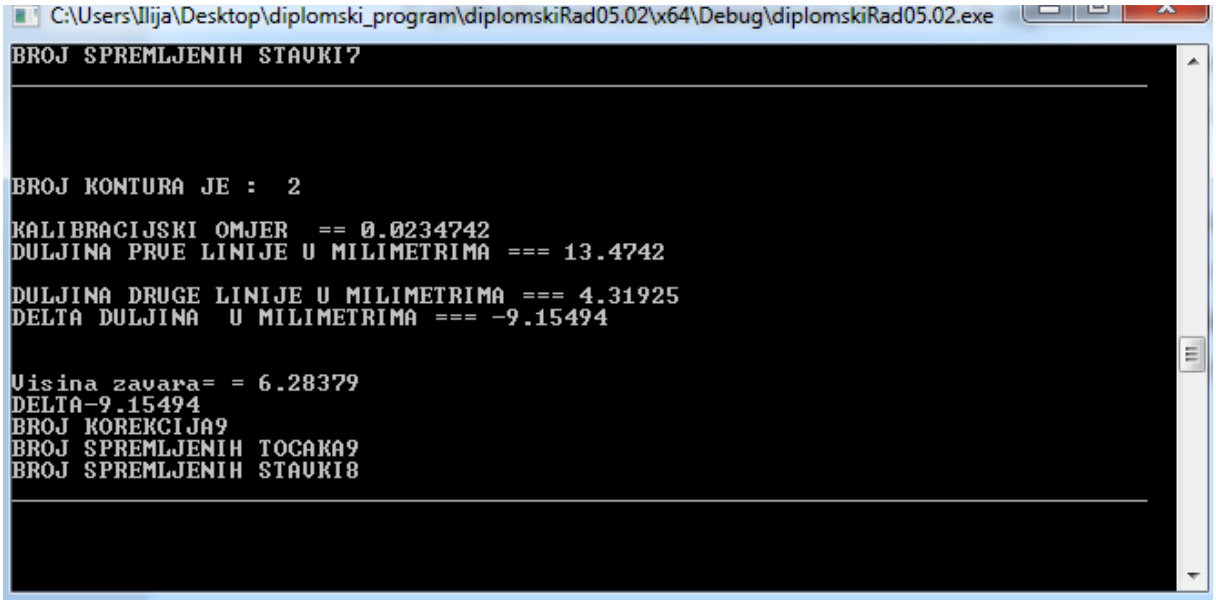
U pozadini se provodi računanje odstupanja na način da se računaju duljine kraće i duže konture nakon čega se korekcija računa tako da se te dvije vrijednosti oduzimaju i dijele s dva.



Slika 33: Računanje korekcije

U prethodno navedenom primjeru na slici 33 to bi izgledalo ovako: (A-B)/2. Međutim, treba biti oprezan i paziti na predznak jer, ukoliko se kriva vrijednost pošalje, robot će napraviti korekciju u pogrešnom smjeru, odnosno, vrhom alata će otići dalje od mjesta zavara.

Uz to, u trenutku kad se počinje računati korekcija pozicije, pokreće se potprogram za dohvaćanje trenutne pozicije robota tako da se od robota dobiva novi set točaka u zapisu [x,y,z,Rx,Ry,Rz]. Taj set točaka je onaj dio putanje koji se treba ponoviti uz korekciju u drugom prolazu. Tokom snimanja na računalu je prikazan prozorčić s vrijednostima prikupljenih podataka kao što je to prikazano na slici 34.



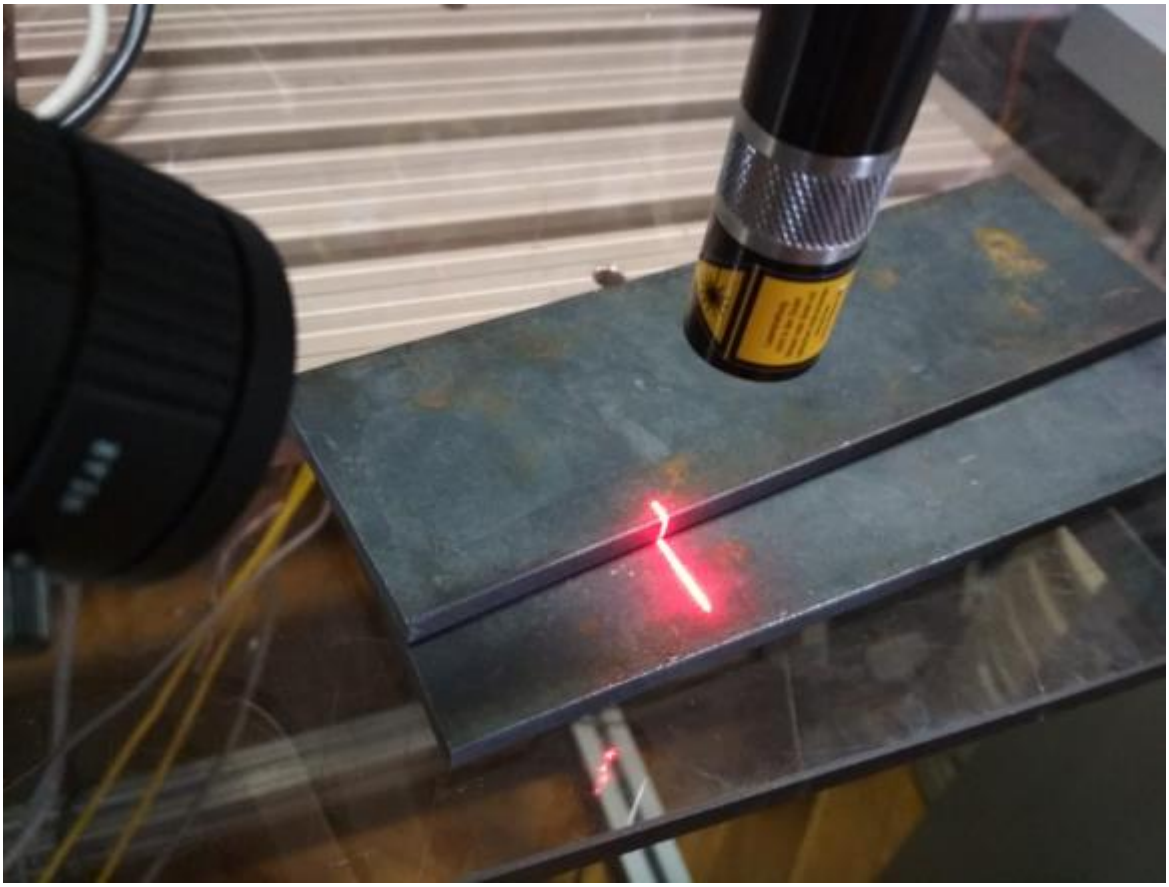
Slika 34: Ispis vrijednosti tokom snimanja

Po završetku snimanja, robot se vraća u prvu snimljenu poziciju i napravi korekciju te ide točku po točku snimljenom putanjom korigirajući svaku poziciju. Nakon ovoga može se promijeniti radni komad s istom putanjom zavarivanja koji ne mora biti obrađen ili postavljen idealno jer će robot napraviti korekciju za dužinu laserske zrake koja je u ovom radu procijenjena na 10mm .

4. Odabirom ovog režima rada provodi se kalibracija gornje kamere. O ovome je bilo riječi u poglavlju 3.2 pa se čitatelj upućuje na to poglavlje. Računalni kod i korištene funkcije u ovom modu rada će biti priloženi kao dodatak ovom radu.

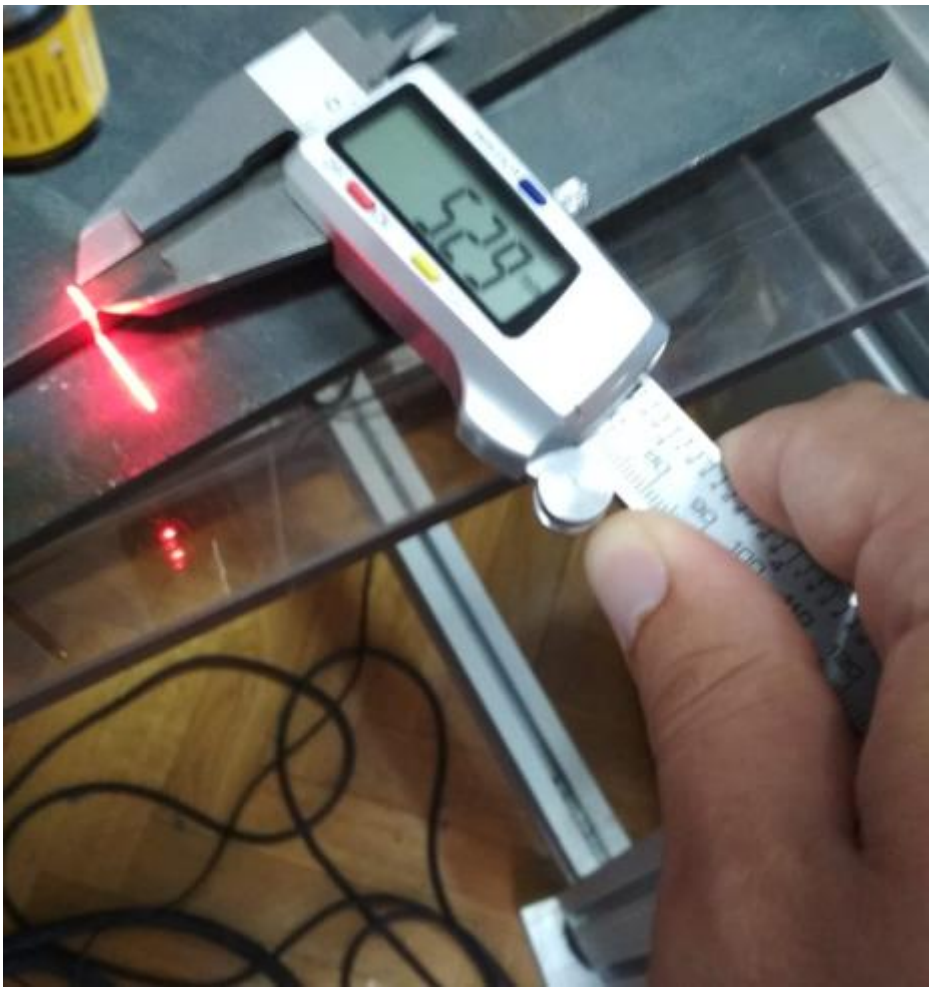
5. EKSPERIMENTALNA PROVJERA TOČNOSTI

U ovom poglavlju prikazat će se jedan primjer korekcije pozicije lasera. Pokretanjem programa dohvaća se slika s gornje kamere te se proizvoljna pozicija iznad predmeta snimanja. Nakon obrade točaka i računanja robotske trajektorije robot se upućuje iznad predmeta snimanja. Dolaskom do zadane pozicije može se vidjeti kako vrh alata nije pozicioniran idealno u odnosu na predmet snimanja što zorno prikazuje slika 35.



Slika 35: Pozicija na kojoj je robot zaustavljen

Pomičnim mjerilom je izmjerena duljina kraće linije, uz to treba uzeti u obzir da zbog fizičke nemogućnosti mjerenje pomičnim mjerilom nije izvedeno na ispravan način tako da je očekivana greška mjerenja veća.



Slika 36: Duljina gornje (kraće) laserske zrake

Izmjerena vrijednost A je 5.29 mm. Istom tehnikom izmjerit će se i duljina duže u ovom slučaju donje laserske linije što prikazuje slika 38. Vrijednosti tih dvaju izmjera bit će uspoređene s vrijednostima koje su dobivene iz računalne analize slike s kamere koja snima konturu zavara. Po završetku računalnog procesiranja računalo će izračunati i korekciju koju je potrebno napraviti da se vrh alata postavi u ispravan položaj u odnosu na predmet snimanja.

Slika 37: Duljina donje (dulje) laserske zrake

Vrijednost očitana s pomičnog mjerila za dulju lasersku zraku jer 13.59 mm. Nadalje, može se izmjeriti i debljina predmeta zavarivanja. Na slici 38 se vidi očitanje za debljinu predmeta zavarivanja. Nakon ponovnog pokretanja programa za korekciju putanje dobivamo rezultate dobivene računalnim vidom.

Slika 38: Mjerenje debljine predmeta snimanja

Nakon ponovnog pokretanja programa za korekciju putanje dobivamo rezultate dobivene računalnim vidom, rezultati su ispisani u prozoru vidljivom na slici 39.

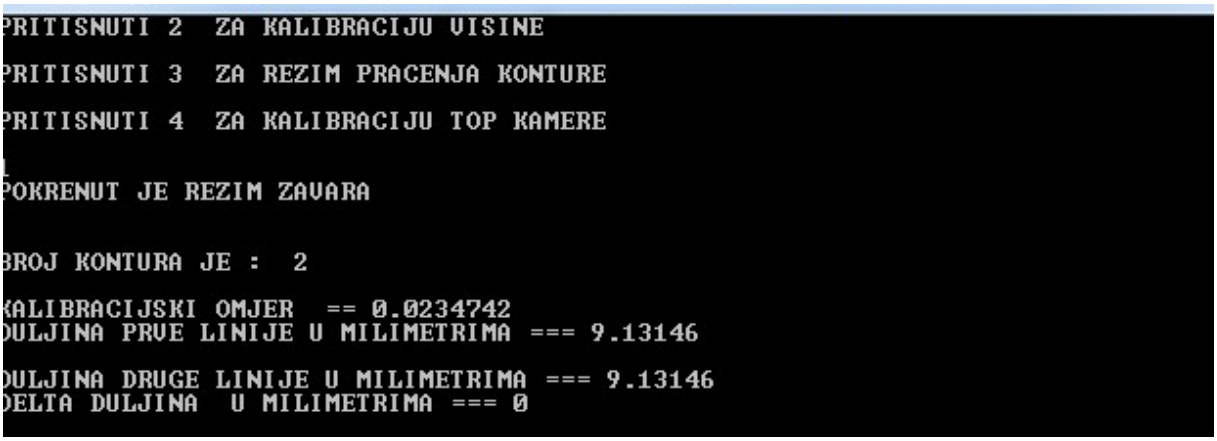
```
BROJ KONTURA JE :  3


BROJ KONTURA JE :  2

KALIBRACIJSKI OMJER  == 0.0234742
DULJINA PRVE LINIJE U MILIMETRIMA === 5.04695

DULJINA DRUGE LINIJE U MILIMETRIMA === 13.0517
DELTA DULJINA  U MILIMETRIMA === 8.0047


Visina zavara= = 6.31423
DELTA8.0047
BROJ KOREKCIJA1
BROJ SPREMLJENIH TOCAKA1
BROJ SPREMLJENIH STAVKI0
```

Slika 39: Rezultati dobiveni računalnim vidom

Tablica 6:Rezultati

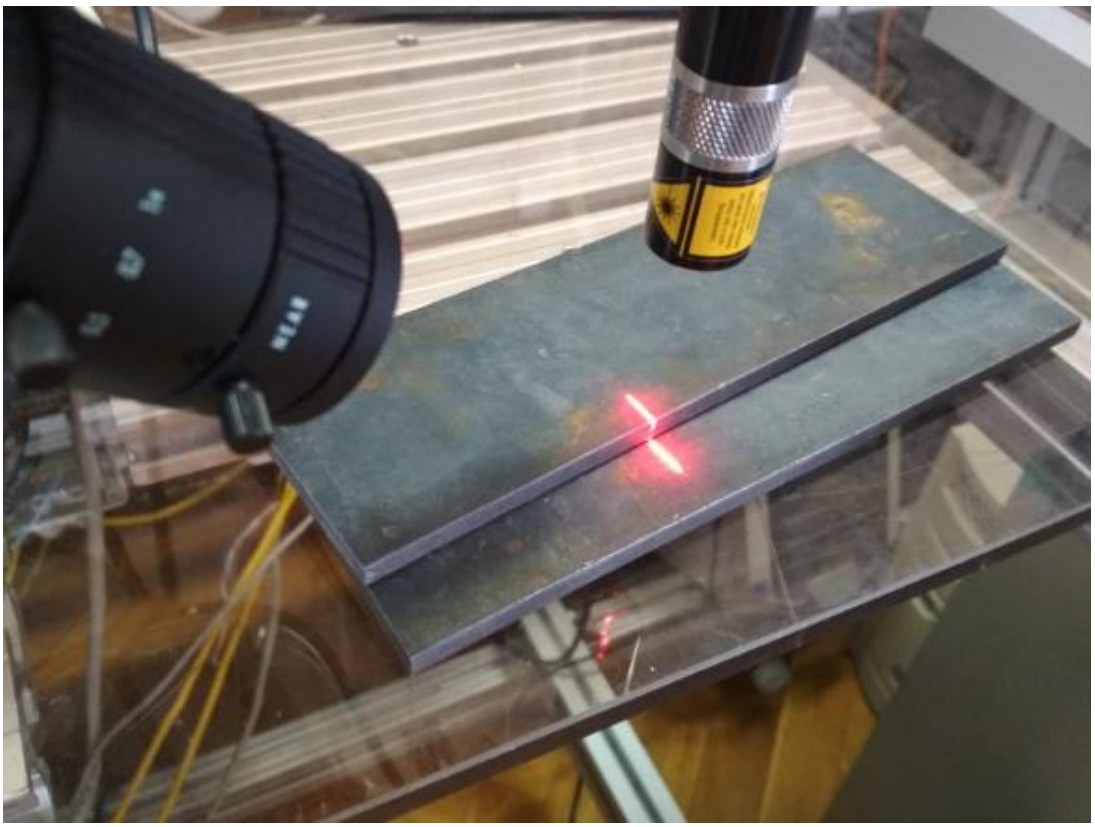
	Izmjerena vrijednost	**Izračunata vrijednost**	**Greška(apsolutna)**
Duljina duže linije	13,59mm	13,05mm	0,54mm
Duljina kraće linije	5,29mm	5,05mm	0,24mm
Visina	6.24mm	6.31mm	0,07mm

Tablica 6 prikazuje usporedbu rezultata mjerenja pomičnim mjerilom i onoga što se dobije računski. Može se primijetiti kako je pogreška nešto veća kod mjerenja duljine laserskih linija, što je i razumljivo, zbog loše i neispravne tehnike mjerenja. Program će nakon što napravi izračun izvesti korekciju pogreške te, ukoliko se nakon toga ponovno pokrene isti program, obje linije dobivaju jednake dimenzije, što dokazuje slika 40.



Slika 40: Rezultati računalnog vida nakon korekcije

Na slici 41 može se vidjeti nova pozicija lasera koja je centrirana u odnosu na onu sa slike 35.



Slika 41: Pozicija nakon korekcije

6. ZAKLJUČAK

U ovom radu je obrađen problem korekcije putanje vrha alata na temelju slike konture zavara. Industrijska oprema korištena u ovom radu integrirana je u jednu cjelinu koja je u interakciji s čovjekom daje željene rezultate. Ideja za rad se razvila s namjerom rješavanja problema zavarivanja radnih komada koji nisu idealno obrađeni i kod kojih postoje blage varijacije u dimenzijama te kao takvi nisu pogodni za robotsko zavarivanje.

U radu je razvijen program koji uz korisnikovu interakciju robotu kreira opću trajektoriju te se robot po njoj kreće i vrši snimanje konture. Kamera i laser koji se nalaze u produžetku robotske ruke prelaze iznad konture zavara te na taj način analizom vizijskih značajki određuju korekciju trajektorije. Za uspješno savladavanje problema bilo je potrebno uspostaviti TCP/IP komunikaciju između svih komponenti sustava te slika dobivenih od dvije industrijske kamere izvući potrebne značajke te ih kalibrirati kako bi se obavila određena mjerenja. Kako bi postojeći sustav bio još točniji poželjna je zamjena kamere koja gleda konturu zavara s kamerom većeg *frame rate*-a.

Kao budući rad vidim nadogradnju sustava na način da se izbaci korisničko kreiranje trajektorije te da se robot samostalno navodi preko konture nakon čega radi korekciju iste.

LITERATURA

[1] UR5 manual

[2] https://z-laser.com/produkt/lasermodul/zm-laser-familie/zm18/#tab_specification

[3] http://repozitorij.fsb.hr/8910/3/Cicvaric_2018_diplomski.pdf

[4] https://opencv.org/about/

[5] https://www.baslerweb.com/en/products/software/basler-pylon-camera-software-suite/

[6] Learning OpenCV by Gary Bradski and Adrian Kaehler

[7] Structuring Elements

[8] Structural Analysis and Shape Descriptors

[9] Camera calibration With OpenCV

[10] Kirić, Nikola Primjena 3D vizijskog sustava za praćenje objekata robotom u realnom vremenu, 2015.

[11] Remote Control Via TCP/IP

[12] Yin, S., Ren, Y., Zhu, J., Yang, S., Ye, S.: A Vision-Based Self-Calibration Method

[13] The URScript Programming Language , V3.10 , Svibanj 2019.

PRILOZI

I. CD-R disc

II. Programski kod

III. Tehnička dokumentacija