Robotsko izuzimanje objekata temeljem boje i oblika

Sinjeri, Luka

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Luka Sinjeri

Zagreb, 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentor:

Doc. dr. sc. Tomislav Stipančić

Student:

Luka Sinjeri

Zagreb, 2022.

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

Zahvaljujem mentoru doc. dr. sc. Tomislavu Stipančiću na zadavanju zadatka i kontinuiranom praćenju rada te asistentu Leonu Korenu, mag. ing. mech. na savjetima koji su mi pomogli u izradi ovog diplomskog rada.

Također zahvaljujem svojoj obitelji, djevojci Magdaleni i prijateljima koji su mi bili velika podrška tijekom studija.

Luka Sinjeri

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

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

DIPLOMSKI ZADATAK

Student: **LUKA SINJERI** Mat. br.: 0035212349

Naslov rada na hrvatskom jeziku: **Robotsko izuzimanje objekata temeljem boje i oblika**

Naslov rada na engleskom jeziku: **Robotic separation of objects based on color and shape**

Opis zadatka:

Računalni modeli temeljeni na algoritmima umjetne inteligencije omogućavaju automatsko prepoznavanje objekata koristeći vizijske senzore smještene u stvarnu okolinu. Prepoznavanje je moguće temeljiti na različitim značajkama objekata te na različitim metodama iz područja strojnog vida. Razvijenu programsku podršku je potom moguće koristiti u sklopu različitih primjena u robotici.

U okviru završnog rada je potrebno:

- Proučiti metode i algoritme za detekciju objekata u stvarnom vremenu.
- Upoznati se s OpenCV programskom bibliotekom otvorenog koda te koristiti prikladne metode u svrhu prepoznavanja objekata na temelju boje i oblika.
- Proučiti karakteristike šest-osnog kolaborativnog robota UR5 te oblikovati i izraditi prikladnu robotsku hvataljku koristeći aditivne tehnologije. Povezati vizijski senzor s robotom.
- Razviti odgovarajuću programsku podršku koja omogućuje funkcionalnosti robotskog izuzimanja objekata temeljem boje i oblika.
- Analizirati učinkovitost razvijenog rješenja u ovisnosti o trenutnim svjetlosnim uvjetima u prostoriji.

Razvijenu robotsku primjenu je potrebno ostvariti koristeći opremu u sklopu Laboratorija za projektiranje izradbenih i montažnih sustava.
U radu je potrebno navesti korištenu literaturu te eventualno dobivenu pomoć.

Zadatak zadan:	Rok predaje rada:	Predviđeni datum obrane:
29. rujna 2022.	1. prosinca 2022.	12. prosinca do 16. prosinca 2022.

Zadatak zadao:
doc. dr. sc. Tomislav Stipančić

Predsjednica Povjerenstva:
prof. dr. sc. Biserka Runje

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

POPIS TEHNIČKE DOKUMENTACIJE

BROJ CRTEŽA	**Naziv iz sastavnice**
LS-DR-01	Nastavak hvataljke
LS-DR-02	Prsti hvataljke

POPIS OZNAKA

Oznaka	Mjerna jedinica	Opis oznake
φ	rad	kut
α	°	kut
X_w, Y_w, Z_w		koordinate 3D točke u globalnom koordinatnom sustavu
u, v		koordinate 2D piksela na ravnini slike
P		projekcijska matrica kamere
K		intrinzična matrica kamere
R		rotacija koordinatnog sustava kamere
t		translacija koordinatnog sustava kamere
f_x, f_y	mm	žarišna duljina kamere
c_x, c_y		udaljenost senzora kamere od optičkog centra

POPIS KRATICA

Kratica	Opis
OpenCV	*Open Source Computer Vision* – knjižnica računalnog vida i strojnog učenja
TCP	*Transmission Control Protocol* – standard koji definira uspostavu veze i prenošenje podataka između servera i klijenta
RGB	*Red Green Blue* – model boja koji se sastoji od crvene, zelene i plave boje
CMY	*Cyan Magenta Yellow* – model boja koji se sastoji od cijan, magenta i žute boje
CMYK	*Cyan Magenta Yellow Black* – model boja koji se sastoji od cijan, magenta, žute i crne boje
HSL	*Hue Saturation Lightness* – model boja definiran faktorima nijanse, zasićenosti i svjetline
HSV	*Hue Saturation Value* – model boja definiran faktorima nijanse, zasićenosti i vrijednosti svjetlosti
FDM	*Fused deposition modeling* – aditivna metoda 3D ispisivanja koja koristi kontinuirani filament termoplastičnog materijala
PLA	*Polylactic acid* – polilaktična kiselina
CAD	*Computer Aided Design* – računalom potpomognuto oblikovanje

SAŽETAK

Razvojem umjetne inteligencije uz sve veću dostupnost potrebne računalne opreme omogućeno je automatizirano prepoznavanje objekata korištenjem vizijskih senzora smještenih u stvarnu okolinu. Zadatak ovog diplomskog rada je razviti programsku podršku koja će omogućiti robotsko izuzimanje temeljeno na značajkama boje i oblika predmeta korištenjem određenih metoda iz područja strojnog vida. Korišteni vizijski senzor za snimanje predmeta iz okoline u stvarnom vremenu je web kamera. Prepoznavanje predmeta omogućeno je putem programa napravljenog korištenjem algoritama OpenCV knjižnice implementirane kroz programski jezik Python. Koordinate pronađenog predmeta šalju se na kolaborativnog robota Universal Robots UR5 koji vrši izuzimanje predmeta traženih karakteristika. Komunikacija između računala i robota ostvaruje se TCP protokolom. Robotska hvataljka potrebna za rukovanje predmetom oblikovana je i izrađena korištenjem aditivne tehnologije. Na početku rada dan je uvod u navedeno područje nakon čega se opisuju koraci izrade programskog rješenja. Na kraju rada analiziraju se dobiveni rezultati, nakon čega se izvodi odgovarajući zaključak.

Ključne riječi: prepoznavanje objekata, robotsko izuzimanje, OpenCV, UR5

SUMMARY

Continuous development of artificial intelligence alongside increased accessibility of required computer equipment provides automated object detection by using visual sensors located in the real environment. Goal of this master's thesis is to develop program support which would enable robotic separation based on attributes of color and shape of the object by utilizing certain methods of machine vision. Visual sensor used to record objects in real time located in environment is web camera. Object detection is achieved by program based on the algorithms that are part of OpenCV library implemented in Python programming language. Coordinates of detected object are sent to collaborative robot Universal Robots UR5 in order to perform robotic separation on object of required attributes. Communication between computer and robot is achieved by using TCP protocol. Robotic gripper needed for manipulation with objects is modeled and constructed by using additive manufacturing. Introduction into mentioned field is given at the beginning of thesis which is followed by describing steps of creating program support. After analyzing results, valid conclusion is given at the end of thesis.

Key words: object detection, robotic separation, OpenCV, UR5

1. UVOD

Razlikovati plavu boju od crvene ili kocku od kugle za ljude je vrlo jednostavan zadatak. Čovjek vrlo lako može prepoznati objekt traženih karakteristika i izdvojiti ga iz skupa različitih objekata. Problem nastaje kada se ovaj vrlo jednostavan zadatak pokuša automatizirati. Rješenje takvog problema pronalazi se upotrebom raznih metoda i algoritama umjetne inteligencije. Računalni modeli temeljeni na algoritmima umjetne inteligencije omogućavaju automatsko prepoznavanje objekata koristeći vizijske senzore smještene u stvarnu okolinu. Područje umjetne inteligencije koje omogućava računalima interpretaciju informacija sa slike naziva se računalni vid.

1.1. Računalni vid

Računalni vid [1] je područje koje se svakodnevno razvija, a posvećeno je analiziranju, izmjeni i visokoj razini interpretacije slike. Cilj računalnog vida je odrediti što se događa ispred objektiva kamere i iskoristiti tu informaciju kako bi se upravljalo određenim sustavom ili kako bi se umjesto originalne slike stvorile nove, koje su informativnije ili estetski ugodnije. Primjena računalnog vida koristi se kod video nadzora, biometrije, automobilske industrije, fotografiranja, filmske industrije, web pretraživanja, medicine, proširene stvarnosti i mnogih drugih područja. U današnje vrijeme kamere imaju mogućnost automatskog fokusiranja na ljudska lica i fotografiranja u trenutku kada se ljudi nasmiju. Sustavi za optičko prepoznavanje teksta pomažu pretvoriti skenirane dokumente u tekst koji se može obraditi ili iščitati naglas koristeći sintetizator glasa. Automobili nude sustave automatizirane asistencije u vožnji koji pomažu kod parkiranja ili upozoravaju vozače o potencijalno opasnim situacijama. Inteligentni video nadzor ima ključnu ulogu kod sigurnosnog nadgledanja javnih područja. Kako pametni uređaji poput mobitela i tableta uključuju kamere i sve više procesorske snage, potreba za aplikacijama temeljenih na računalnom vidu postaje sve veća. Funkcije kao što su generiranje panoramskog prikaza visoke rezolucije na osnovu nekoliko uzastopnih fotografija ili prepoznavanje QR koda kako bi se dohvatile informacije o pojedinom proizvodu preko Interneta samo su neke od mogućnosti koje računalni vid nudi. Naposlijetku treba definirati razliku između strojnog i računalnog vida. Računalni vid odnosi se na obradu slike u svrhu dobivanja novih informacija sa slike. Strojni vid koristi informacije dobivene obradom slike snimljene vizijskim senzorom za izvršavanje određenih funkcija u industriji. Primjer strojnog vida je automatizirani pregled gotovih proizvoda, dok je složeniji primjer upravljanje robotskim

sustavom koji obuhvaća identifikaciju, pozicioniranje i pravilnu orijentaciju na osnovu informacije dobivene putem senzora.

1.2. Prepoznavanje objekata

Jedno od ključnih područja računalnog vida je prepoznavanje objekata na digitalnoj slici ili videozapisu. Ovo svojstvo omogućuje računalima da „vide“, odnosno prepoznaju određena vizualna svojstva objekata kako bi ih mogla podijeliti u kategorije. Drugim riječima, prepoznavanje objekata omogućuje da računalo odredi gdje se unutar slike pojedini objekt nalazi i kojoj skupini pojedini objekt pripada. Prepoznavanje objekata moguće je izvesti korištenjem metoda obrade slike ili korištenjem dubokog učenja. Metoda obrade slike podrazumijeva izradu nove slike na osnovu učitane, a to se izvodi alatima kao što je OpenCV, koji je opisan u ovom poglavlju. Korištenje dubokog učenja u svrhu prepoznavanja objekata uključuje upotrebu nadziranog učenja. Model vrši kategorizaciju objekata temeljem učenja iz uzoraka dostupnih putem baze podataka. [2]

Neki od algoritama za prepoznavanje objekata u stvarnom vremenu temeljenih na dubokom učenju su:

- R-CNN (eng. *Region-based Convolutional Neural Networks*)
- HOG (eng. *Histogram of Oriented Gradients)*
- SSD (eng. *Single Shot Detector*)
- YOLO (eng. *You Only Look Once*).

Slika 1.1 prikazuje prepoznavanje automobila u stvarnom vremenu putem algoritma za prepoznavanje objekata nastalog korištenjem metoda dubokog učenja.

Slika 1.1. Primjena algoritma za prepoznavanje objekata [2]

1.3. Opis rada

U sklopu ovog rada potrebno je izraditi programsku podršku koja bi omogućila robotsko izuzimanje predmeta korištenjem strojnog vida. Osnovna ideja zadatka je napraviti algoritam koji će pomoću informacije dobivene vizijskim senzorom poslati naredbu robotu gdje se objekt traženih atributa nalazi u svrhu izuzimanja predmeta pomoću robotske hvataljke. Potrebni algoritam za prepoznavanje objekata u stvarnom vremenu temelji se na značajkama boje i oblika predmeta. Stoga su kao predmeti rada u okviru ovog zadatka odabrana geometrijska tijela u nekoliko boja kako bi se pokazala funkcionalnost aplikacije za predmete različitih oblika i različitih boja. Na početku rada opisani su osnovni pojmovi vezani uz korištene metode prepoznavanja. Nakon toga detaljno je pojašnjen cijeli postupak izrade aplikacije dok su na kraju rada dani rezultati u različitim uvjetima korištenja nakon čega je izveden zaključak. Programski jezik korišten u sklopu izrade aplikacije je Python.

1.3.1. Korišteni programski jezik

Python je interpretiran i objektno orijentiran programski jezik. Tvorac Pythona je nizozemski programer Guido van Rossum koji je 1991. izbacio prvu verziju ovog programskog jezika. Python sadrži module, iznimke, dinamičke tipove podataka visoke razine, dinamičko povezivanje i klase. Sučelje podržava mnoge knjižnice i moguće ga je proširiti u C ili C++ programskim jezicima. Podržani operativni sustavi su Microsoft Windows, Linux i macOS. [3] Razlozi zbog kojih je Python odabran za potrebe ovog rada su njegova velika rasprostranjenost, jednostavnost korištenja, širok izbor modula i knjižnica, a povrh svega je i besplatan. Zbog karakteristika kao što su razumljiva sintaksa koju je jednostavno naučiti te velike podrške za knjižnice otvorenog koda, Python je među najkorištenijim programskim jezicima u području strojnog učenja. Verzija korištena za potrebe ovog rada je Python 3.10., a za testiranje i prevođenje programskog koda korišten je Microsoft Visual Studio Code.

1.3.2. Korištene knjižnice

Knjižnica je zbirka unaprijed napisanih kodova koju je moguće koristiti iterativnim putem u svrhu smanjenja vremena potrebnog za programiranje. Najveće i najčešće korištene knjižnice unutar Pythona su TensorFlow, Keras, SciPy, Numpy, OpenCV itd. U sklopu ovog rada korištene su knjižnice OpenCV, Numpy i urx koje će biti ukratko opisane u nastavku.

OpenCV (eng. *Open Source Computer Vision*) najveća je i najpopularnija knjižnica računalnog vida i strojnog učenja otvorenog koda na svijetu. Knjižnica obuhvaća preko 2500 algoritama među kojima su klasični i najsuvremeniji algoritmi računalnog vida i strojnog učenja. Algoritmi

se koriste za prepoznavanje lica, prepoznavanje objekata, kategorizaciju ljudskih pokreta, praćenje položaja kamere, praćenje objekata u pokretu, stvaranje 3D modela iz objekata, stvaranje 3D oblaka točaka putem stereo kamere, spajanje slika u svrhu dobivanja slike visoke rezolucije cijelog kadra, uklanjanje efekta crvenih očiju s fotografija, praćenje kretanja očiju itd. OpenCV broji više od 45 tisuća aktivnih korisnika u zajednici. Koriste ju velike tvrtke među kojima su Microsoft, Google, Sony itd. Sadrži sučelja za većinu najpopularnijih programskih jezika među kojima su Python, C++, Java i MATLAB, a podržava operacijske sustave Windows, Linux i macOS. Slika 1.2 prikazuje logo OpenCV-a. [4]

Slika 1.2. OpenCV [4]

Numpy je Python knjižnica koja se koristi za rad s velikim i višedimenzionalnim poljima i matricama. Također obuhvaća mnoge matematičke funkcije visoke razine pomoću kojih se izvršavaju operacije na složenim poljima i matricama. [5]

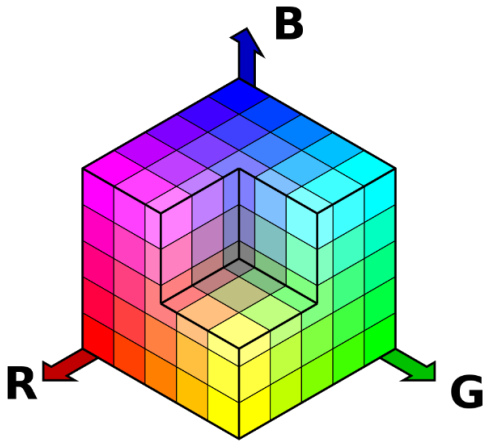
Knjižnica urx koristi se za kontroliranje robota proizvođača Universal Robots. Zamišljena je kao modul za jednostavno kontroliranje robota kod postupaka izuzimanja predmeta, ali također se može koristiti za robotsko zavarivanje i druge robotske primjene temeljene na senzorima koje ne zahtijevaju visoku kontrolnu frekvenciju. [6]

2. PREPOZNAVANJE TEMELJEM BOJE

Model boja [7] je apstraktni matematički model koji opisuje način na koji boje mogu biti prikazane kao niz brojeva, obično putem tri ili četiri vrijednosti ili elementa boje. Kada je ovaj model povezan s preciznim opisom interpretacije elemenata uzimajući u obzir vizualnu percepciju rezultirajući skup naziva se prostorom boja. U nastavku je opisano kako je moguće modelirati spektar boja vidljivih ljudima pomoću nekoliko modela koji se najčešće koriste. Modeli boja podijeljeni su u dvije osnovne skupine. Prva skupina sadrži modele koji se temelje na dodavanju i oduzimanju pojedine boje. U ovu skupinu pripadaju modeli boja RGB (eng. *Red*, *Green*, *Blue*), koji se koristi u monitorima te CMY (eng. *Cyan*, *Magenta*, *Yellow*) i CMYK (eng. *Cyan*, *Magenta*, *Yellow*, *Black*), koji se primjenjuju na uređajima koji služe za nanošenje boje na papir. U drugu skupinu pripadaju modeli cilindričnih koordinata, a ključni modeli iz te skupine su HSL (eng. *Hue*, *Saturation*, *Lightness*) i HSV (eng. *Hue*, *Saturation*, *Value*).

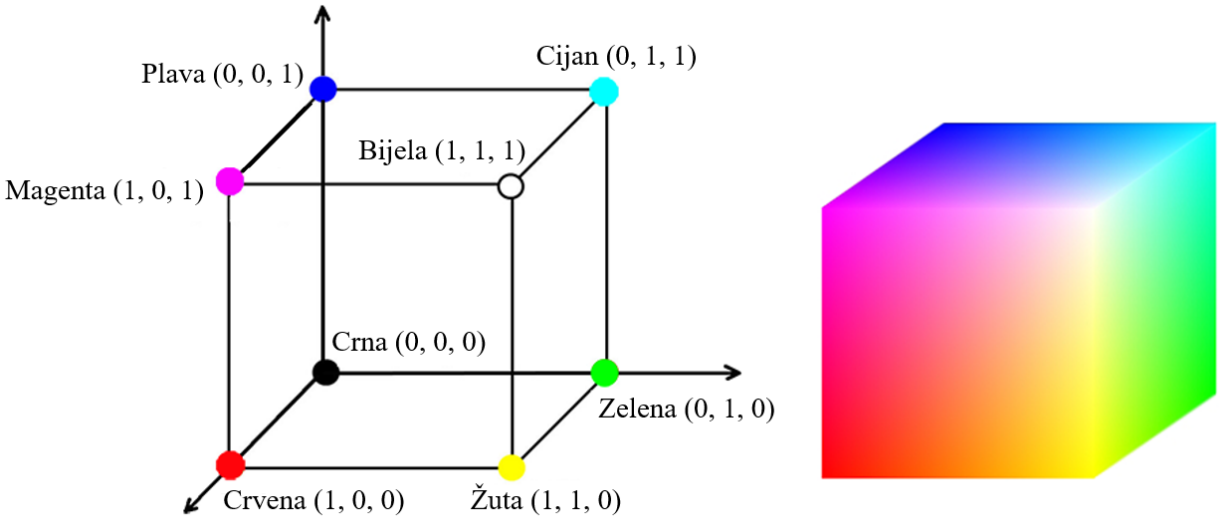
2.1. RGB

RGB model [8] je aditivan model boja koji koristi primarne boje crvenu, zelenu i plavu pomiješane na različite načine kako bi se prikazao širok spektar drugih boja. Korištenje drugih primarnih boja načelno je moguće, ali kombinacijom crvene, zelene i plave pokriven je najširi raspon ljudskog vidnog spektra. Glavna uloga RGB modela je opažanje i prikazivanje slike na elektroničkim uređajima kao što su računala i televizija, a također se koristi u fotografiranju. Ovaj model jako ovisi o uređaju na kojem se koristi što znači da će različiti uređaji za iste vrijednosti boja prepoznati ili prikazati drugačiju nijansu pojedine boje. Uređaji na kojima se RGB najčešće koristi su televizije, monitori, mobilni telefoni, video projektori itd.



Slika 2.1. RGB model [8]

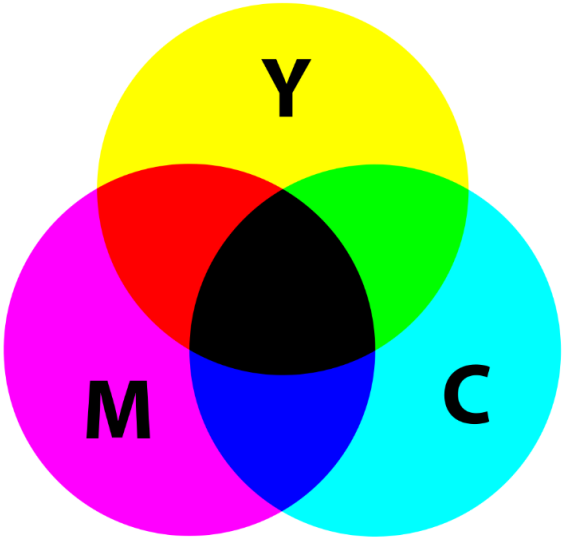
Slika 2.1 prikazuje grafički prikaz RGB modela. Model je opisan u pravokutnom koordinatnom sustavu i svaku boju moguće je zapisati korištenjem tri vrijednosti (R, G, B) koje se odnose na pojedine vrijednosti crvene (eng. *red*), zelene (eng. *green*) i plave (eng. *blue*) svjetlosti. Te tri komponente mogu poprimiti vrijednost počevši od nule do neke maksimalne vrijednosti u ovisnosti o odabranom načinu brojčanog označavanja. Na taj način bilo koja boja nalazi se unutar kocke čije dužine stranica odgovaraju najvećoj vrijednosti pojedine boje. Tri koordinatne osi koje se nalaze u ishodištu kocke označavaju nijanse crvene, zelene i plave, koje su poznate kao primarne boje. Os x predstavlja crvenu boju, os y zelenu boju te os z predstavlja plavu boju. Vrijednost se može iskazati korištenjem decimalnih ili cijelih brojeva raspona 2^n.[8] Slika 2.2 prikazuje kocku s ucrtanim vrijednostima pojedinih koordinata. Vrh kocke smješten u ishodište koordinatnog sustava s koordinatama (0, 0, 0) predstavlja crnu boju najveće zasićenosti. Nasuprot ishodišta nalazi se točka zapisana koordinatama (1, 1, 1) i ta točka opisuje bijelu boju. Dijagonala koja spaja crnu i bijelu boju definirana je različitim vrijednostima sive boje (eng. *grayscale*) s obzirom da su duž dijagonale vrijednosti sva tri parametra jednake (R = G = B). Na slici se može vidjeti kako se crvenu boju može prikazati sa (1, 0, 0), dok se primjerice žuta boja prikazuje sa (1, 1, 0).



Slika 2.2. RGB kocka [9]

2.2. CMYK

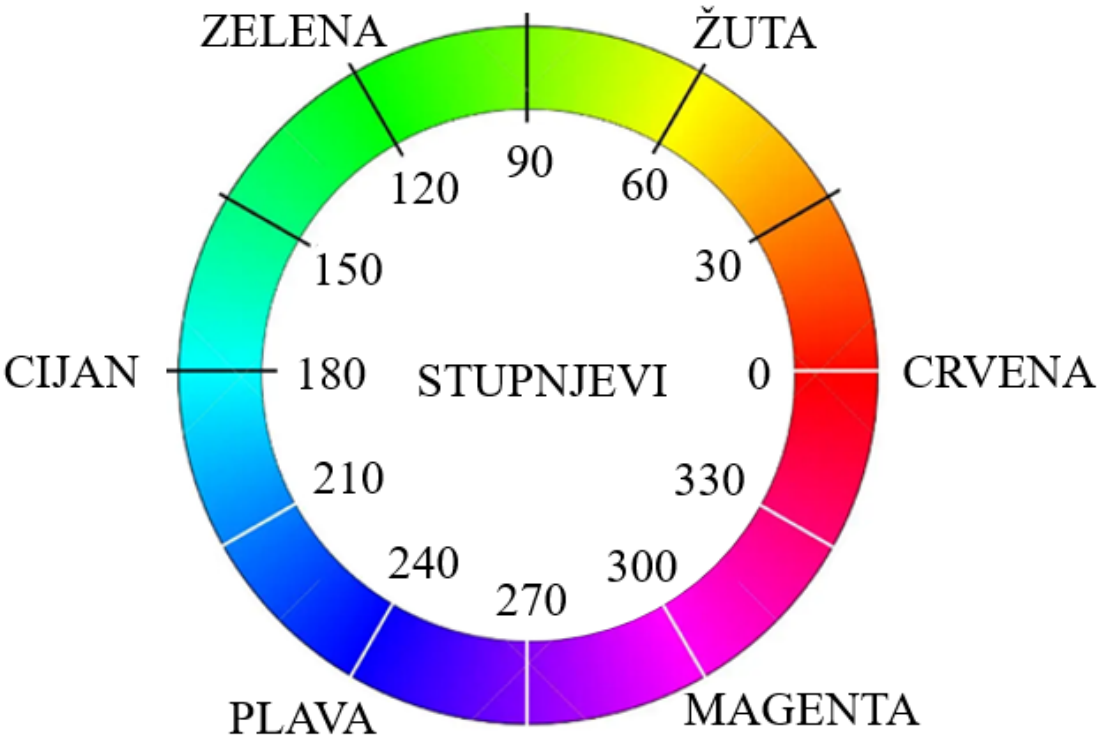
CMYK model boja [10] koristi suptraktivnu sintezu za razliku od aditivne metode korištene kod prethodno opisanog modela i upotrebljava se za tiskanje boje na papir. Prostor boja kod ovog modela uži je u odnosu na onaj koji posjeduje prethodno opisani RGB model što rezultira time da uređaj za tiskanje ne može stvoriti sve boje koje se nalaze na traženom predmetu za ispis prikazanom preko računala. Ideja je da se RGB primarnim bojama osvijetli bijeli papir. Spoj sve tri primarne svjetlosti kao rezultat daje bijelu boju. Ukoliko se pomiješa crvena svjetlost zajedno s plavom svjetlosti dobije se boja koja nosi naziv magenta (eng. *magenta*). Spoj crvene zajedno sa zelenom daje žutu boju (eng. *yellow*), dok se spajanjem zelene i plave dobiva boja zvana cijan (eng. *cyan*). Na taj način dobiveni su osnovni suptraktivni elementi CMY modela po kojima model i nosi ime. Već spomenuta suptraktivna sinteza korištena kod ovog modela podrazumijeva oduzimanje boja iz bijele što je razlika u odnosu na dodavanje boja u crnu u slučaju prethodnog modela. Cijan se dobiva oduzimanjem crvene svjetlosti iz bijele svjetlosti, magenta se dobiva oduzimanjem zelene svjetlosti iz bijele, dok se žuta dobiva oduzimanjem plave svjetlosti iz bijele. Rezultat spajanja sva tri elementa CMY modela dat će crnu boju zbog apsorpcije cjelokupne svjetlosti. No na taj način ne dobije se pravilan izgled crne boje, a također dolazi do velikog navlaživanja papira što produljuje period sušenja. Kako bi se riješio taj problem CMY modelu pridodaje se četvrti element u obliku crne boje. Iz razloga što se slovo B već koristi za plavu boju, dogovorom je utvrđeno da se za crnu boju koristi slovo K odakle proizlazi naziv CMYK. Slika 2.3 prikazuje grafički izgled modela.



Slika 2.3. CMYK model [10]

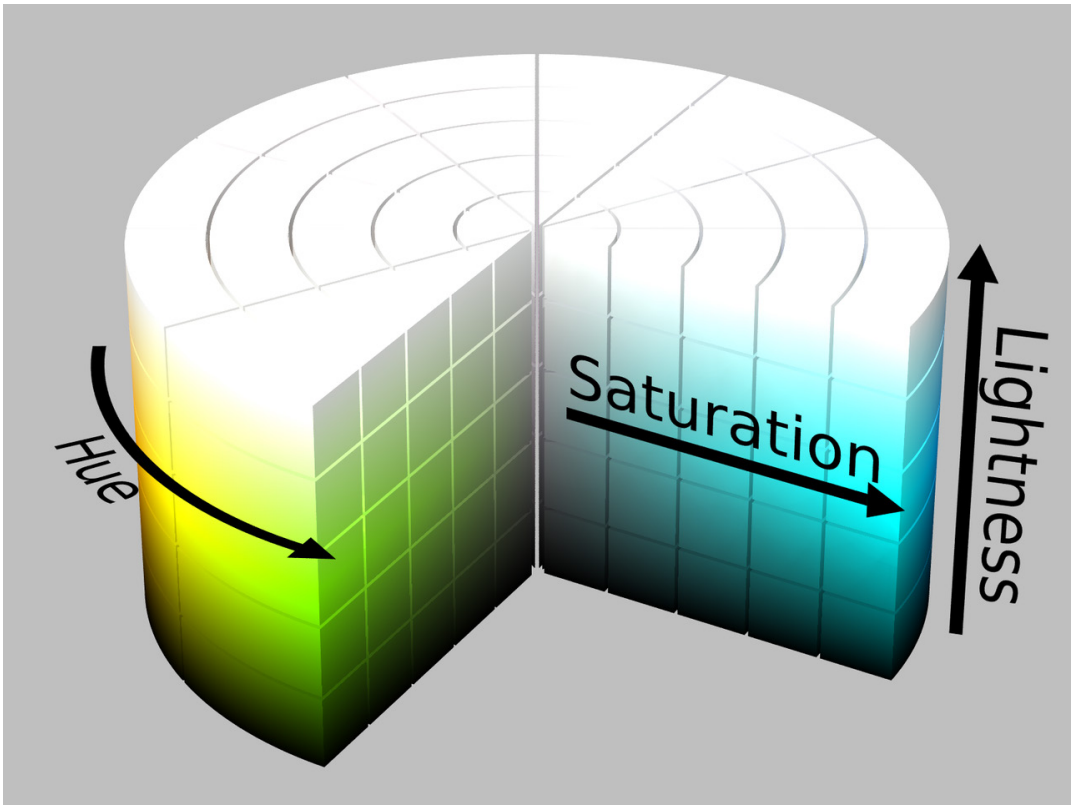
2.3. HSL

Kod HSL modela boja je opisana s tri elementa po kojima je model dobio ime, a to su nijansa boje, zasićenost i svjetlina. Slovo H označava nijansu (eng. *hue*) boje, a definirana je položajem na kružnoj paleti boja poput one prikazane na slici (Slika 2.4). Položaj se izražava kutom između ishodišta koje se nalazi kod crvene boje i bilo koje promatrane boje. Računa se u suprotnom smjeru od onog kod kazaljki na satu. Ishodišni kut od 0° označava crvenu boju, dok se na 120° nalazi zelena boja nakon koje slijedi plava na 240° i naposlijetku paleta završava na 360° s crvenom bojom. Slovo S opisuje zasićenost (eng. *saturation*), a definirana je odmakom od središta kružne palete. Područja uz rub kružnice imaju maksimalnu zasićenost dok su u centru vrijednosti zasićenosti najniže. Kako bi se iskazala zasićenost koristi se vrijednost u postocima pri čemu 100% označava maksimalnu zasićenost pojedine boje. Zadnje slovo ovoga modela L označava intenzitet svjetlosti reflektirane od površine (eng. *lightness*).



Slika 2.4. Paleta boja [11]

Slika 2.5 prikazuje cilindar koji se koristi za grafički prikaz HSL modela. Približavanjem gornjem vrhu cilindra svjetlina kružne palete boja se povećava, dok će približavanje donjoj stranici cilindra dovesti do smanjenja svjetline, tj. povećanja tame. Gornja površina cilindra ima vrijednost 1 i označava bijelu boju dok donja stranica cilindra ima vrijednost 0 i predstavlja crnu boju. [12]

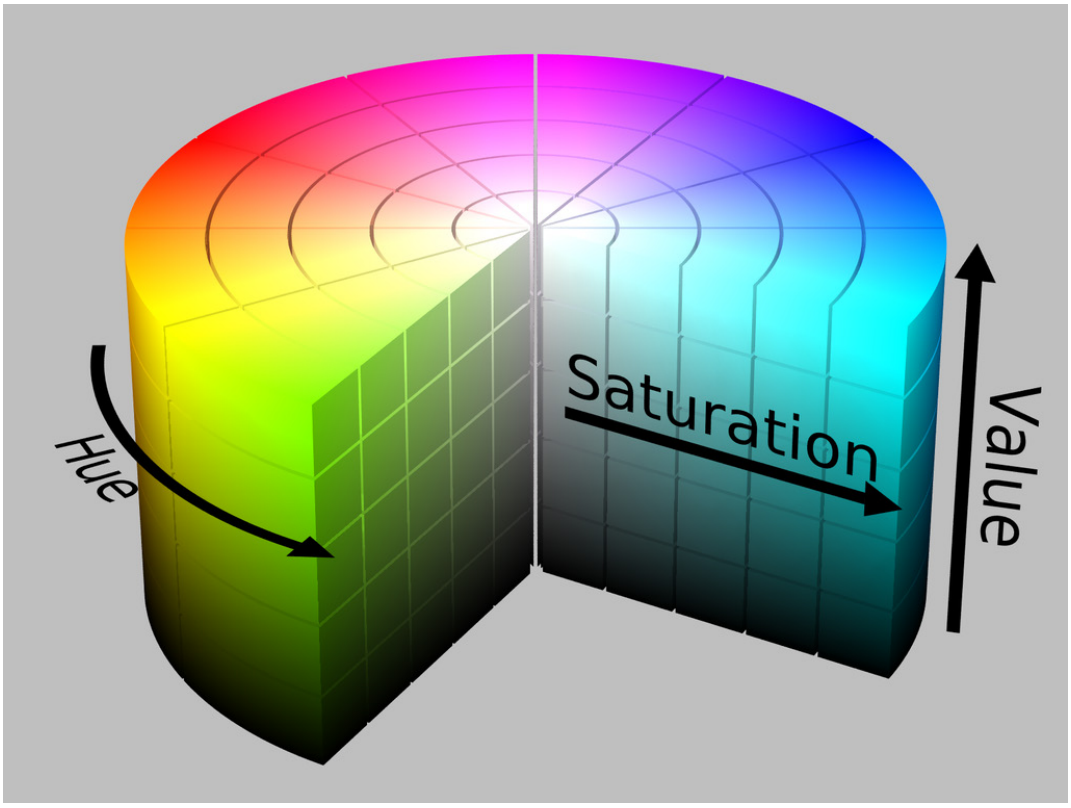


Slika 2.5. HSL model [12]

Kružnu paletu boja moguće je predočiti kao kružnu površinu dobivenu vodoravnim presijecanjem cilindra u nekoj točki. Smještajem komplementarnih boja jedne nasuprot drugoj cijeli sustav čini jednostavnijim i intuitivnijim od prethodno spomenutog RGB modela.

2.4. HSV

Razlika između HSL i HSV modela je u tome što potonji prikazuje kako se boja ponaša pod svjetlom. Kod HSL modela maksimalna svjetlina rezultirala je s potpuno bijelom bojom, dok kod HSV modela boja s maksimalnom vrijednošću svjetline odgovara osvjetljavanju bijelog svijetla na obojani predmet. Primjer za to je osvjetljenje crvenog predmeta svijetlim bijelim svjetlom što za rezultat daje predmet koji je i dalje crvene boje, samo što je svjetliji i intenzivniji, dok je kod osvjetljenja prigušenim svjetlom taj isti objekt i dalje crven, ali ovog puta tamniji. Kao i u slučaju prethodno opisanog modela slovo H označava nijansu, S označava zasićenje, a V se odnosi na vrijednost svjetline (eng. *value*). [12] Slika 2.6 prikazuje cilindar koji služi kao grafički prikaz HSV modela.



Slika 2.6. HSV model [12]

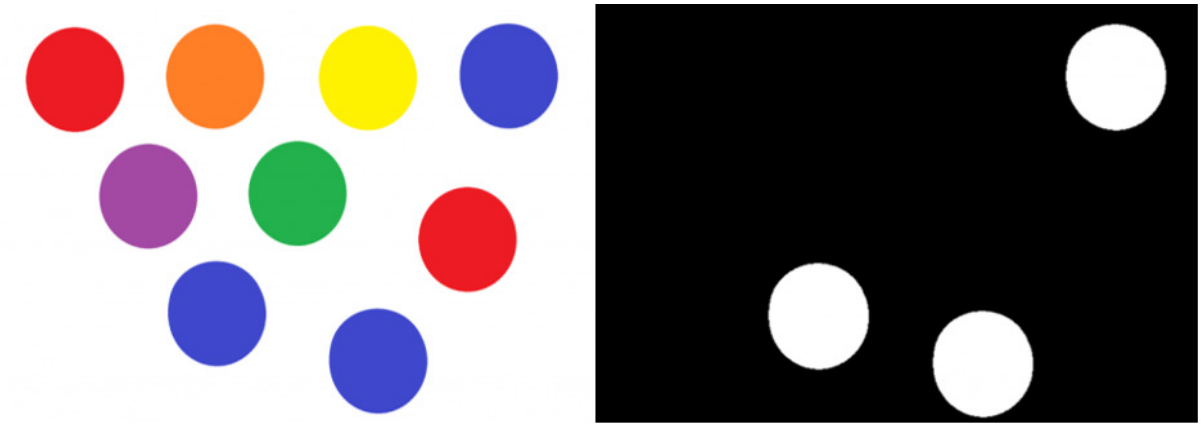
Nijansa boje određuje se na isti način kao i kod prethodno opisanog modela. Zasićenost raste od centra kružne palete prema rubovima, dok vrijednost svjetline raste od donje stranice valjka gdje su tamniji tonovi prema gornjoj stranici gdje su najsvjetliji tonovi. HSV model korišten je u okviru programskog rješenja kako bi se pomoću intervala pronađenih vrijednosti napravila maska putem koje bi se izolirali traženi predmeti određenih boja.

3. PREPOZNAVANJE TEMELJEM OBLIKA

Osnovni zadatak kod prepoznavanja objekata putem oblika u stvarnom vremenu je izdvajanje traženog objekta od ostatka prizora. Metoda pomoću koje se pojedini objekti odvajaju od ostatka slike zove se segmentacija. Postoje razne metode segmentacije, a u sklopu ovog rada korištena je metoda koja se pokazala brzom i pouzdanom u stvarnom vremenu, a to je metoda maskiranja.

3.1. Maskiranje

Maskiranje je metoda koja se temelji na izdvajanju dijela prikaza koji odgovara zadanim parametrima. U ovom konkretnom slučaju uvjeti izdvajanja dani su preko određene nijanse tražene boje. U prijevodu, na slici će se maskirati sva područja koja ne odgovaraju traženoj boji čime se na slici prikazuju samo objekti od interesa kako bi se na njima provodila daljnja analiza. Maskiranje je jako korisno u uvjetima gdje je pozadina nepromjenjive boje i razlikuje se od predmeta rada. Slika 3.1 prikazuje rezultat maskiranja unutar OpenCV knjižnice. U konkretnom slučaju lijeva strana slike prikazuje ulazne parametre, dok je s desne strane dostupan prikaz traženih objekata nakon maskiranja koji su u ovom slučaju svi objekti plave boje. Detaljan postupak izrade maske bit će opisan u idućem poglavlju.



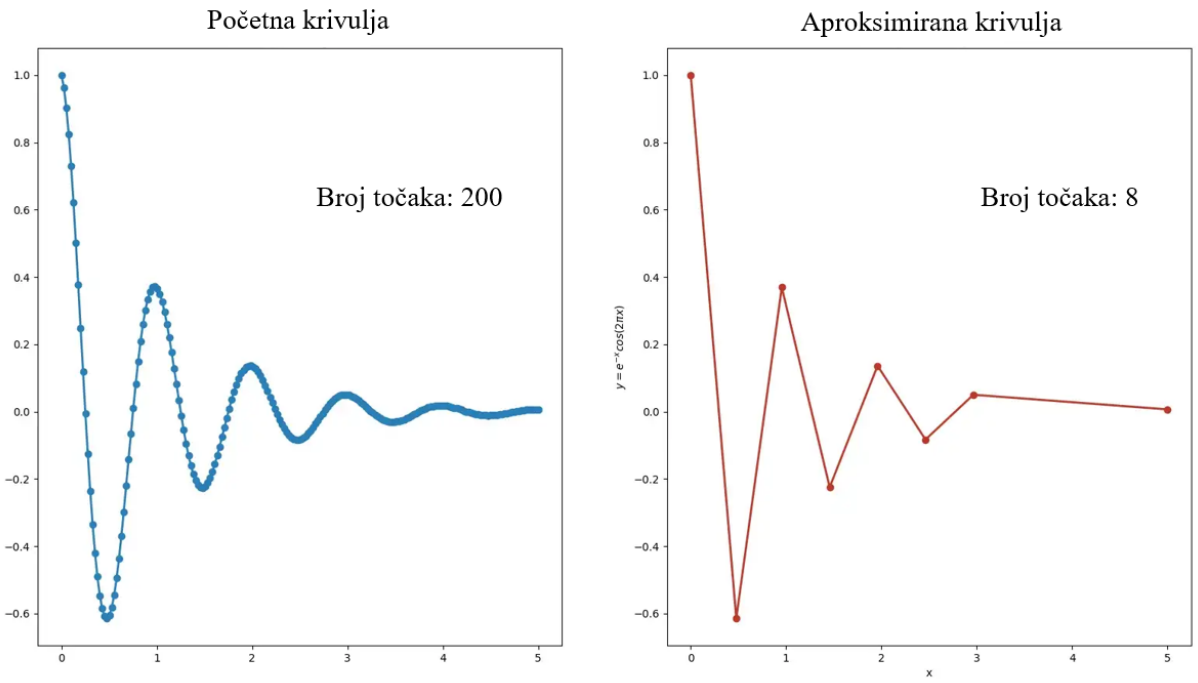
Slika 3.1. Maskiranje [13]

3.2. Aproksimacija kontura

Metoda kojom se određena krivulja prikazuje pomoću nove krivulje s manjim brojem međusobno povezanih točaka naziva se aproksimacija kontura. Unutar OpenCV-a postoji funkcija pomoću koje se aproksimiraju poligonalne krivulje s proizvoljnom točnošću. Funkcija se poziva pomoću naredbe cv2.approxPolyDP, a sastoji se od tri argumenta, a to su *curve*, *epsilon* i *closed*. Kao prvi argument navodi se kontura koju se želi aproksimirati. *Epsilon*

označava maksimalnu udaljenost između ulazne krivulje i aproksimirane krivulje i tim parametrom se određuje točnost aproksimacije. Zadnji argument *closed* označava da li je krivulja zatvorena ili otvorena. Funkcija se temelji na Ramer–Douglas–Peuckerovom algoritmu koji početnu krivulju pojednostavi na krivulju sastavljenu od manje segmenata. Novonastala kontura je određena na osnovu maksimalne udaljenosti između ulazne i aproksimirane krivulje, koja se također naziva Hausdorffova udaljenost. [14]

Rezultat je krivulja koja se sastoji od podskupa točaka koje su opisivale početnu krivulju. Slika 3.2 prikazuje aproksimaciju krivulje određenim parametrom *epsilon*. Ova funkcija bit će korištena nakon primjene maske kako bi se utvrdio oblik objekta temeljem broja pronađenih kontura.



Slika 3.2. Aproksimacija krivulje [15]

4. IZRADA APLIKACIJE

U okviru ovog rada zadana je izrada programske aplikacije pomoću koje se omogućuje robotsko izuzimanje objekata na osnovu kriterija boje i oblika. Programski jezik korišten za izradu aplikacije je Python, unutar kojega je potrebno učitati knjižnice OpenCV i Numpy, korištene za prepoznavanje objekata te također knjižnicu naziva urx pomoću koje se ostvaruje kontrola nad robotom UR5 putem kojega je izvršeno izuzimanje objekata. Cijeli postupak izrade programske podrške sastoji se od niza koraka koji će redom biti objašnjeni unutar ovog poglavlja.

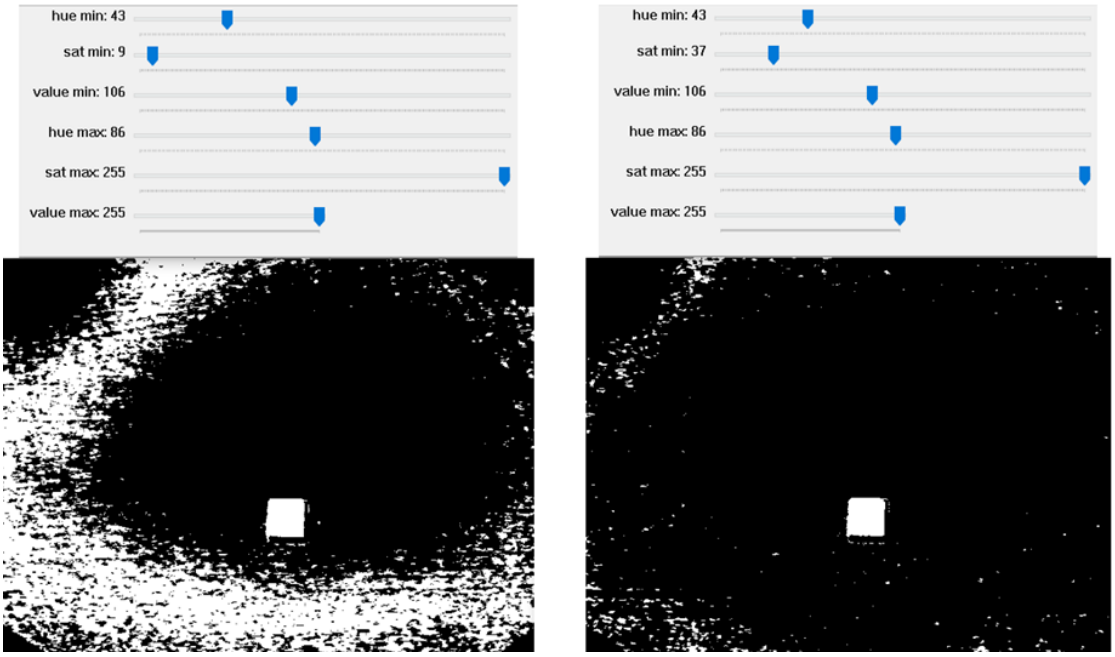
4.1. Izrada maske

Prvi korak u izradi aplikacije bio je određivanje odgovarajuće maske koja bi omogućila izdvajanje traženih objekata od ostatka radnog prostora. Maskiranje unutar OpenCV-a provodi se putem HSV modela boje postavljanjem odgovarajućih intervala nijanse, zasićenosti i vrijednosti svjetline za pojedinu boju. Na ovaj način postavljeni su intervali za nekoliko korištenih boja, ali taj spektar može se jednostavno proširiti jer je opisanu metodu moguće koristiti za definiranje maske bilo koje proizvoljne boje. Jedina mana ovog modela je velika ovisnost o svjetlosnim uvjetima, međutim dokle god je izvor svjetlosti konstantnog intenziteta, metoda će pružati zadovoljavajuće rezultate. S obzirom da je slika unutar OpenCV-a po zadanim postavkama prikazana pomoću BGR modela boja, koji nije ništa drugo nego standardni RGB model s promijenjenim redoslijedom boja potrebno je provesti pretvorbu slike u HSV model što se izvršava pomoću funkcije cv2.COLOR_BGR2HSV. Kako bi se proizvoljno birale pojedine vrijednosti na osnovu zadanih objekata različitih boja kreiran je program koji se sastoji od alatne trake na kojoj su zadane minimalne i maksimalne vrijednosti parametara nijanse, zasićenosti i svjetlosti koje OpenCV prepoznaje. Mijenjanjem parametara putem alatne trake dobivaju se različiti intervali potrebni za prepoznavanje različitih boja. Konačan prikaz maske dobiven je pomoću funkcije cv2.inRange koja kao argumente uzima ulaznu sliku pretvorenu u HSV model boja, niz od tri vrijednosti koje predstavljaju minimalne vrijednosti nijanse, zasićenosti i svjetlosti te naposlijetku niz koji definira maksimalne vrijednosti ta tri parametra. Nizovi su kreirani pomoću funkcije np.array. Slika 4.1 prikazuje linije koda kojima je napravljen program za određivanje maske bilo koje proizvoljne boje.

```
import cv2
import numpy as np

def nothing(x):
    pass

cap = cv2.VideoCapture(1)
cv2.namedWindow("Trackbars")
cv2.resizeWindow("Trackbars", 600, 300)

cv2.createTrackbar("hue min", "Trackbars", 0, 179, nothing)
cv2.createTrackbar("sat min", "Trackbars", 0, 255, nothing)
cv2.createTrackbar("value min", "Trackbars", 0, 255, nothing)
cv2.createTrackbar("hue max", "Trackbars", 179, 179, nothing)
cv2.createTrackbar("sat max", "Trackbars", 255, 255, nothing)
cv2.createTrackbar("value max", "Trackbars", 255, 255, nothing)

while True:
    _, frame = cap.read()
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    l_h = cv2.getTrackbarPos("hue min", "Trackbars")
    l_s = cv2.getTrackbarPos("sat min", "Trackbars")
    l_v = cv2.getTrackbarPos("value min", "Trackbars")
    u_h = cv2.getTrackbarPos("hue max", "Trackbars")
    u_s = cv2.getTrackbarPos("sat max", "Trackbars")
    u_v = cv2.getTrackbarPos("value max", "Trackbars")

    lower = np.array([l_h, l_s, l_v])
    upper = np.array([u_h, u_s, u_v])
    mask = cv2.inRange(hsv, lower, upper)

    cv2.imshow("frame", frame)
    cv2.imshow("mask", mask)

    if cv2.waitKey(20) & 0xFF==ord('d'):
        break
cap.release()
cv2.destroyAllWindows
```

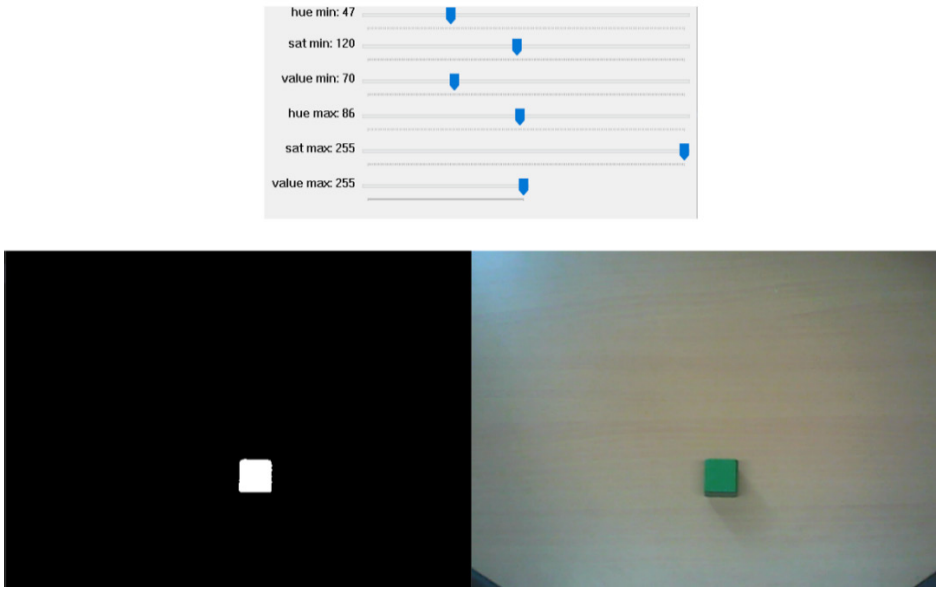
Slika 4.1. Program za određivanje maske

U nastavku je prikazan postupak postavljanja donje i gornje granice nijanse, zasićenosti i svjetlosti za zelenu boju. Slika 4.2 prikazuje kako se izgled maske mijenja u ovisnosti o promjeni faktora zasićenosti.



Slika 4.2. Postavljanje maske

Slika 4.3 prikazuje vrijednosti koje rezultiraju potrebnom maskom s lijeve strane, dok se ulazna slika nalazi na desnoj strani. Nakon što postavljene vrijednosti rezultiraju odgovarajućom maskom na kojoj se vidi samo predmet odabrane boje i ništa drugo, zapisujemo interval maksimalnih i minimalnih vrijednosti nijanse, zasićenosti i svjetlosti kako bi se mogli koristiti za daljnju analizu predmeta.



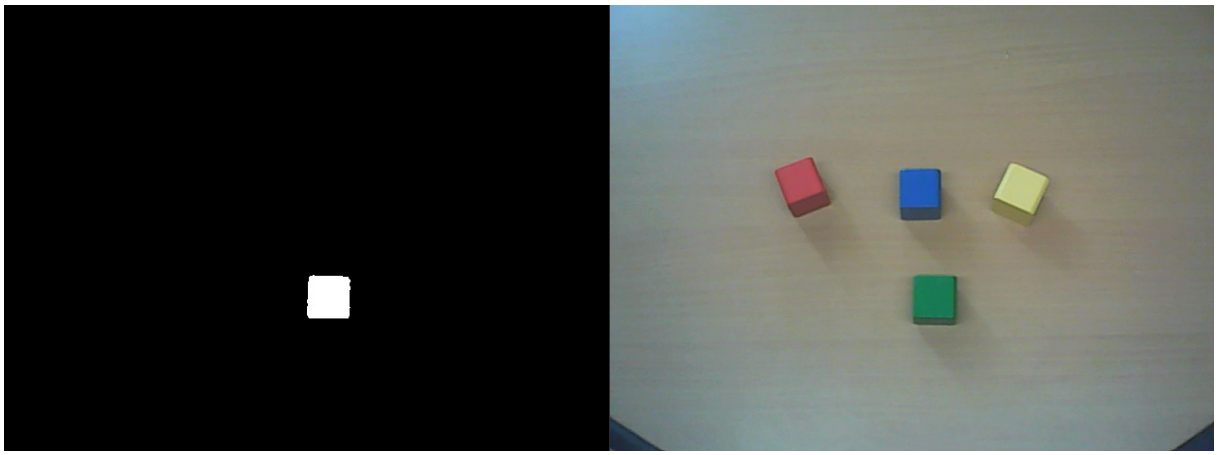
Slika 4.3. Vrijednosti postavljene maske

Isti proces traženja maske ponovljen je za sve boje korištene u sklopu ovog rada, a njihove vrijednosti dane su u tablici (Tablica 4.1).

Tablica 4.1. Vrijednosti maske pojedinih boja

	Crvena	Plava	Žuta	Zelena
Nijansa min.	0	101	14	65
Zasićenost min.	96	92	83	101
Svjetlost min.	136	117	186	92
Nijansa max.	179	122	35	83
Zasićenost max.	255	255	255	255
Svjetlost max.	255	255	255	255

Slika 4.4 prikazuje kako maska pravilnih vrijednosti funkcionira na način da prikriva sve boje osim tražene, u konkretnom slučaju, zelene boje.



Slika 4.4. Maska zelene kocke

4.2. Određivanje pozicije predmeta

Idući korak u izradi programske aplikacije je pronaći točnu koordinatu predmeta na slici kako bi se ta informacija u kasnijim koracima mogla prenijeti na robota. Za početak treba pronaći konture na slici, a to se postiže pozivom funkcije cv2.findContours kojoj se kao ulazni argument zadaje maska dobivena u prethodnom koraku. Na taj način funkcija će kao rezultat odrediti krivulju koja spaja sve točke duž ruba područja iste boje, odnosno intenziteta. U ovom slučaju to će biti područje skupa bijelih točaka na crnoj pozadini. Nakon toga potrebno je pronaći težište zadanog predmeta. Matematički gledano težište nekog oblika je aritmetički prosjek n točaka od kojih se taj oblik sastoji, gdje je prva točka x_1, a krajnja točka x_n što je prikazano jednadžbom (1). [16]

$$c = \frac{1}{\mathrm{n}} \sum_{\mathrm{i}=1}^{\mathrm{n}} x_{\mathrm{i}} \tag{1}$$

Težište se može izračunati preko funkcije cv2.moments u koju se putem brojača unose vrijednosti svake točke od koje se sastoji ranije otkriveni oblik. Kao rezultat te funkcije dobije se težinski prosjek intenziteta piksela na slici. Nakon toga traži se prosjek vrijednosti po horizontalnoj i vertikalnoj osi što konačno daje središte otkrivenog oblika na slici. Formula za izračun težišta prikazana je jednadžbama (2) i (3) pri čemu se c_x odnosi na x koordinatu, c_y označava y koordinatu, dok M označava rezultat funkcije cv2.moments, tj. težinski prosjek intenziteta piksela. Slika 4.5 prikazuje dio koda kojim se izračunava težište predmeta.

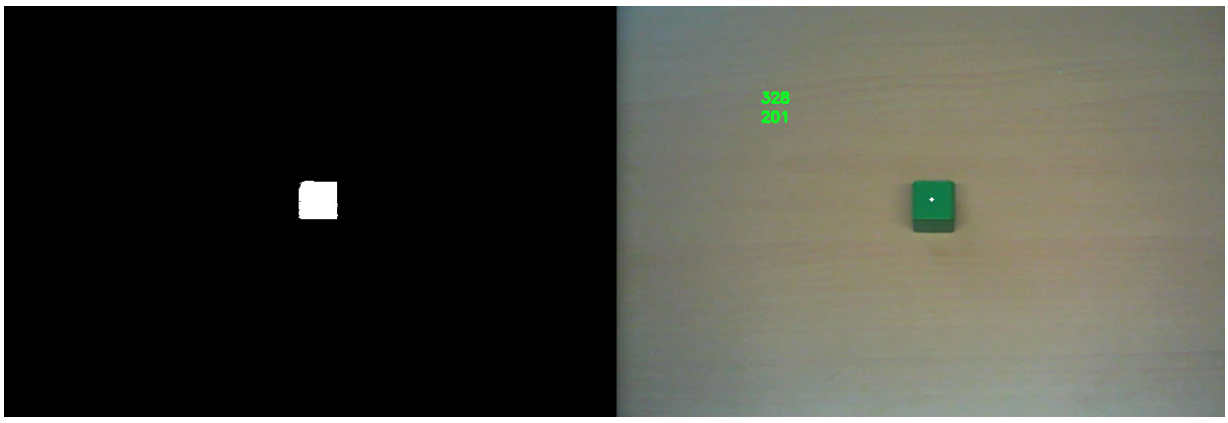
$$c_{\mathrm{x}} = \frac{M_{10}}{M_{00}} \tag{2}$$

$$c_{\mathrm{y}} = \frac{M_{01}}{M_{00}} \tag{3}$$

```
import cv2
import numpy as np

cap = cv2.VideoCapture(0)

while True:
    _, frame = cap.read()
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    lower = np.array([47, 133, 107])
    upper = np.array([86, 255, 255])
    mask = cv2.inRange(hsv, lower, upper)

    cnts, hei = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    for c in cnts:
        area = cv2.contourArea(c)
        if area > 300:

            M=cv2.moments(c)
            cx= int(M["m10"]/M["m00"])
            cy= int(M["m01"]/M["m00"])
            cv2.circle(frame, (cx, cy), 2, (255,255,255), -1)
            cv2.putText(frame, "centar", (cx-20, cy-20), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,255,0),1)
            cv2.putText(frame, str(cx), (150, 100), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (36, 255, 12), 2)
            cv2.putText(frame, str(cy), (150, 120), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (36, 255, 12), 2)

    cv2.imshow("frame", frame)
    cv2.imshow("mask", mask)
    if cv2.waitKey(20) & 0xFF==ord('d'):
        break
cap.release()
cv2.destroyAllWindows()
```

Slika 4.5. Program za izračun težišta

Slika 4.6 prikazuje rezultat pronalaska težišta. Bijela točka na zelenoj kocki služi kao vizualni prikaz težišta kojeg je funkcija dala kao izlaznu vrijednost, a koordinate te točke dane su brojevima ispisanim na desnoj strani slike. U ovom slučaju ti brojevi označuju horizontalni i vertikalni broj piksela na kojima je težište pronađeno i kao takvi nisu pogodni za slanje informacije o koordinatama objekta na robota, stoga će u daljnjim koracima biti nužno izvršiti pretvorbu piksela u milimetre s kojima se može zadati točan položaj gdje robot treba izvršiti izuzimanje predmeta.



Slika 4.6. Pozicija težišta

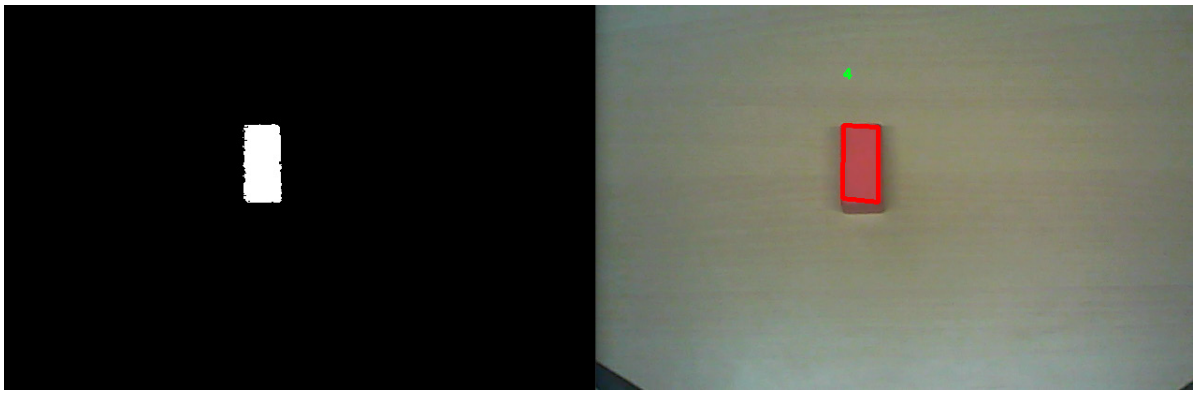
4.3. Određivanje kuta zakreta predmeta

Iduća bitna informacija koju je potrebno prenijeti na robota je iznos kutnog zakreta predmeta u odnosu na neku referentnu os kako bi robot mogao zakrenuti vrh alata i pravilno izvršiti izuzimanje predmeta. S obzirom da je jedan od predmeta rada cilindar kod kojega je kut zakreta nemoguće izračunati potrebno je napraviti izuzeće za taj predmet. To je napravljeno putem if uvjeta tako što je zadan ukupan broj krivulja koje su prethodno aproksimirane funkcijom cv2.approxPolyDP. Slika 4.7 prikazuje dio koda pomoću kojega se izračunava broj aproksimiranih krivulja.

```
import cv2
import numpy as np

cap = cv2.VideoCapture(0)

while True:
    _, frame = cap.read()
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    lower = np.array([0, 137, 146])
    upper = np.array([179, 255, 255])
    mask = cv2.inRange(hsv, lower, upper)

    cnts, hei = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    for c in cnts:
        area = cv2.contourArea(c)
        if area > 300:
            peri=cv2.arcLength(c, True)
            approx=cv2.approxPolyDP(c, 0.02*peri, True)
            x, y, w, h = cv2.boundingRect(c)

            cv2.putText(frame, str(len(approx)), (x, y -50), cv2.FONT_HERSHEY_SIMPLEX,
            0.5, (36, 255, 12), 2)
            cv2.drawContours(frame, [approx], 0, (0,0,255),3)

    cv2.imshow("frame", frame)
    cv2.imshow("mask", mask)

    if cv2.waitKey(20) & 0xFF==ord('d'):
        break
cap.release()
cv2.destroyAllWindows
```

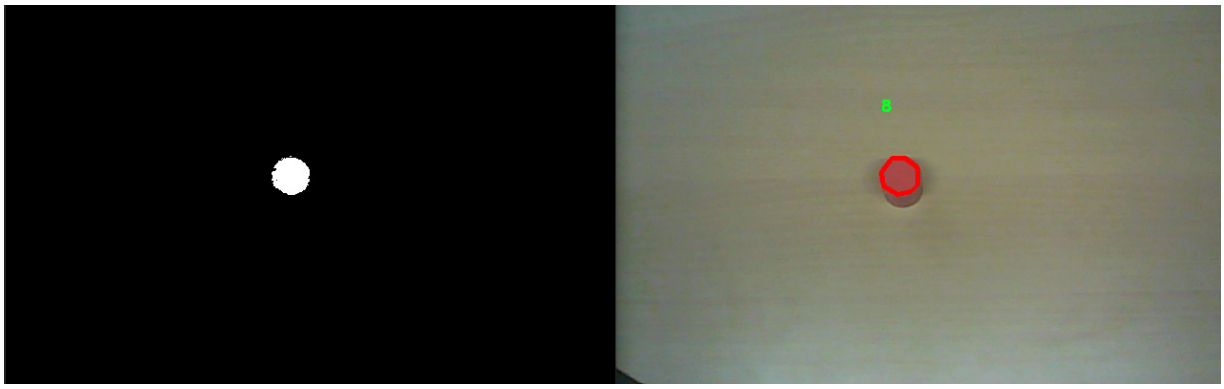
Slika 4.7. Program za izračun broja aproksimiranih krivulja

Slika 4.8 prikazuje broj aproksimiranih krivulja u slučaju kvadra i kocke koji iznosi 4.



Slika 4.8. Aproksimacija kvadra

U slučaju cilindra broj aproksimiranih krivulja kružne površine iznosi dvostruko više od tog iznosa. Slika 4.9 prikazuje aproksimaciju korištenog cilindra.



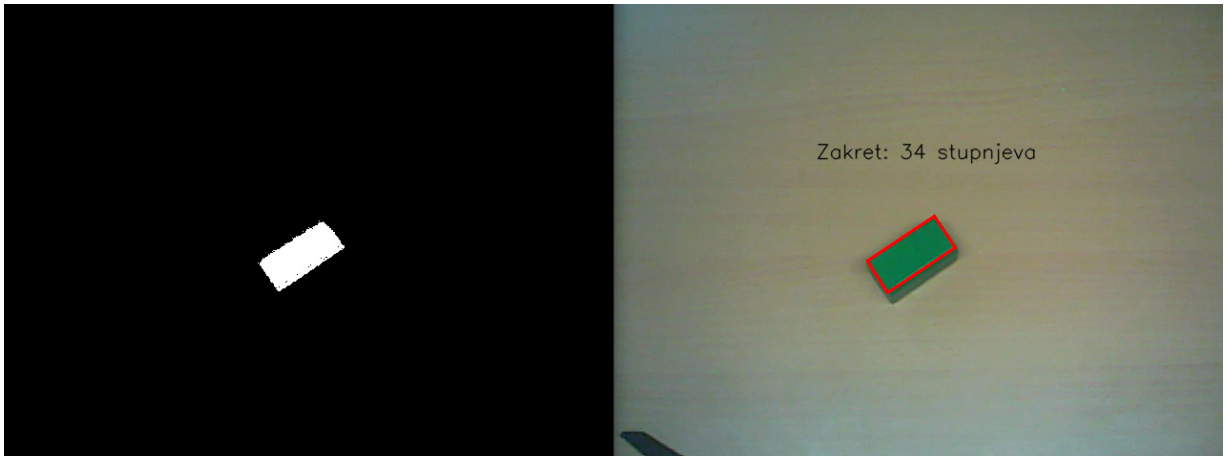
Slika 4.9. Aproksimacija cilindra

Stoga je postavljen uvjet da se izračunavanje kuta zakreta provodi samo ukoliko program pronađe manje od 5 aproksimiranih krivulja. Kako bi se izračunao sam kutni zakret korištena je funkcija cv2.minAreaRect koja pronalazi pravokutnik koji kao ulaznu varijablu uzima skup svih točaka od kojih se sastoji pronađena kontura. Nakon toga uzimaju se ključni parametri dobivenog pravokutnika kao što su središte, širina i visina pomoću kojih se može izračunati kut zakreta. Slika 4.10 prikazuje dio koda koji provodi izračun kuta zakreta.

```
import cv2 as cv
import numpy as np

cap = cv.VideoCapture(0)

while True:
    isTrue, frame = cap.read()
    hsv = cv.cvtColor(frame, cv.COLOR_BGR2HSV)
    lower = np.array([47, 123, 109])
    upper = np.array([82, 255, 255])
    mask = cv.inRange(hsv, lower, upper)

    contours, _ = cv.findContours(mask, cv.RETR_LIST, cv.CHAIN_APPROX_NONE)

    for i, c in enumerate(contours):
        area = cv.contourArea(c)
        if area < 100 or 100000 < area:
            continue

        peri=cv.arcLength(c, True)
        approx=cv.approxPolyDP(c, 0.02*peri, True)
        cv.drawContours(frame, [approx], 0, (0,255,0),3)
        if len(approx) < 5:
            rect = cv.minAreaRect(c)
            box = cv.boxPoints(rect)
            box = np.int0(box)

            center = (int(rect[0][0]),int(rect[0][1]))
            width = int(rect[1][0])
            height = int(rect[1][1])
            angle = int(rect[2])

            if width < height:
                angle = 90 - angle
            else:
                angle = -angle

            label = "Zakret: " + str(angle) + " stupnjeva"
            cv.putText(frame, label, (center[0]-100, center[1]-300),
                cv.FONT_HERSHEY_SIMPLEX, 0.7, (0,0,0), 1, cv.LINE_AA)
            cv.drawContours(frame,[box],0,(0,0,255),2)

    cv.imshow('frame', frame)
    cv.imshow('maska', mask)
    if cv.waitKey(20) & 0xFF==ord('d'):
        break
cap.release()
cv.destroyAllWindows()
```

Slika 4.10. Program za izračun kuta zakreta

Slika 4.11 prikazuje izračunati kut zakrenutog predmeta. OpenCV daje rješenje kutnog zakreta iskazano u stupnjevima, međutim kao takvo nije prikladno za slanje na robota jer knjižnica urx pomoću koje se kontrolira robot preko Pythona napisana je u radijanima i koristi ih kao ulazne parametre za upravljanje zglobovima robota.

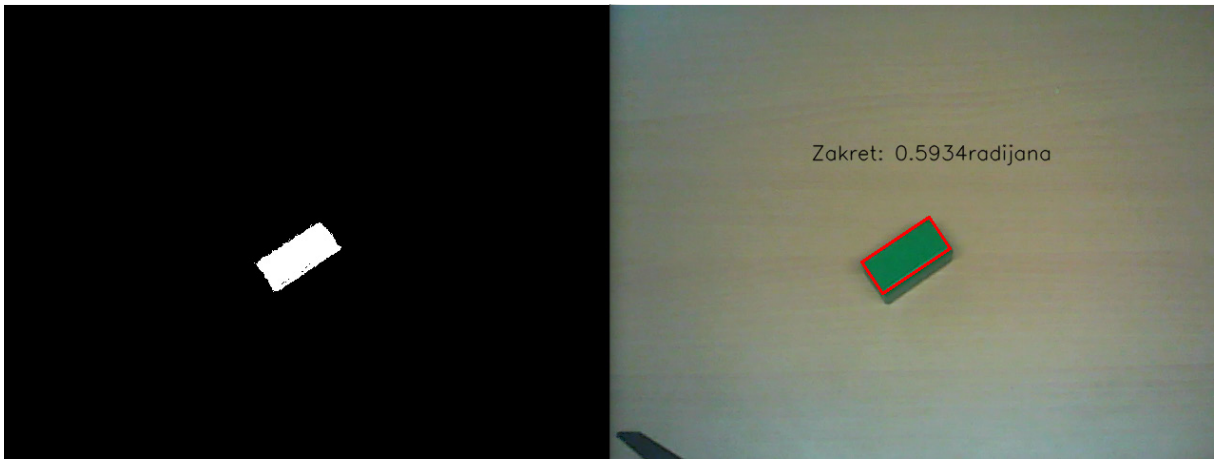


Slika 4.11. Kut zakreta iskazan u stupnjevima

Jednostavnim matematičkim izrazom (4) vrši se pretvorba stupnjeva u radijane, pri čemu je φ kut u radijanima, dok je α kut izražen u stupnjevima.

$$\varphi = \alpha \times \frac{\pi}{180^{\circ}} \tag{4}$$

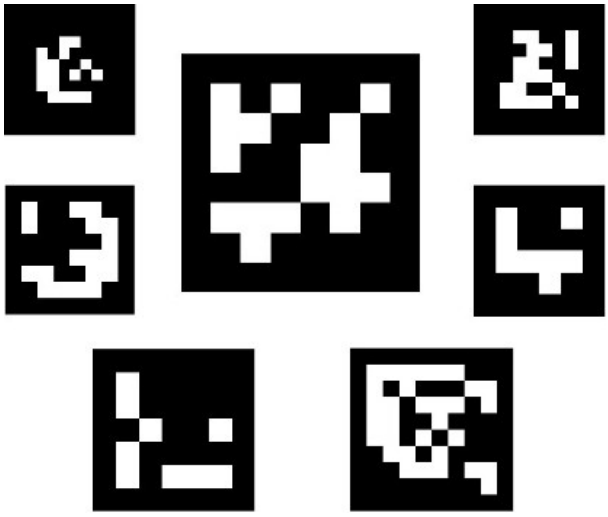
Slika 4.12 prikazuje rezultat nakon pretvorbe te pokazuje isti kutni zakret kao i kod prethodne slike, samo što je ovoga puta vrijednost zakreta iskazana u radijanima, zaokružena na 4 decimalna mjesta pomoću funkcije round.



Slika 4.12. Kut zakreta iskazan u radijanima

4.4. Određivanje ishodišta

Kako bi se koordinate predmeta uspješno prenijele na robota potrebno je definirati koordinatni sustav pomoću kojega je moguće odrediti položaj predmeta u radnom prostoru. Ranije je utvrđeno kako funkcija korištena za pronalazak težišta daje informaciju o položaju objekta u obliku udaljenosti horizontalnih i vertikalnih vrijednosti piksela od početne vrijednosti koja se nalazi u gornjem lijevom kutu slike. Stoga je nužno definirati ishodište na način pogodan za postavljanje u radnu okolinu kako bi se preko njega mogla izračunati udaljenost do predmeta i prenijeti tu informaciju na robota. Rješenje tog problema može se pronaći korištenjem ArUco markera. Slika 4.13 prikazuje primjer nekoliko takvih markera.



Slika 4.13. ArUco markeri [17]

ArUco markeri su binarni kvadrati sastavljeni od mreže crnih i bijelih polja s unaprijed definiranim identifikacijskim značajkama koje je moguće vrlo jednostavno i brzo prepoznati na slici. Osim identifikacije, svakom markeru moguće je odrediti početni rub koji je u ovom slučaju iskorišten kao ishodište koordinatnog sustava. Slika 4.14 prikazuje dio koda koji se koristi za detekciju i prikaz odabranog ruba markera.

```
import cv2
from cv2 import aruco

cap = cv2.VideoCapture(0)

def findAruco(frame, marker_size=6, total_markers=250):
    gray=cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    key=getattr(aruco,f'DICT_{marker_size}X{marker_size}_{total_markers}')
    arucoDict=aruco.Dictionary_get(key)
    arucoParam=aruco.DetectorParameters_create()
    bbox,ids, _ =aruco.detectMarkers(gray,arucoDict,parameters=arucoParam)
    corners, ids, _ = aruco.detectMarkers(gray, arucoDict, parameters=arucoParam)
    if corners:
        x1 = int (corners[0][0][0][0])
        y1 = int (corners[0][0][0][1])
        cv2.putText(frame, str(x1), (100, 250), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (36, 255, 12), 2)
        cv2.putText(frame, str(y1), (100, 270), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (36, 255, 12), 2)
        cv2.circle(frame, (x1, y1), 4, (0,0,255), -1)
    return ids, bbox

while True:
    _,frame=cap.read()
    bbox, ids = findAruco(frame)
    if cv2.waitKey(20) & 0xFF==ord('d'):
        break
    cv2.imshow("frame", frame)
cap.release()
cv2.destroyAllWindows()
```

Slika 4.14. Program za prepoznavanje ArUco markera

Slika 4.15 prikazuje položaj početnog ruba markera označen crvenim krugom, a njegov položaj ispisan je putem programa.

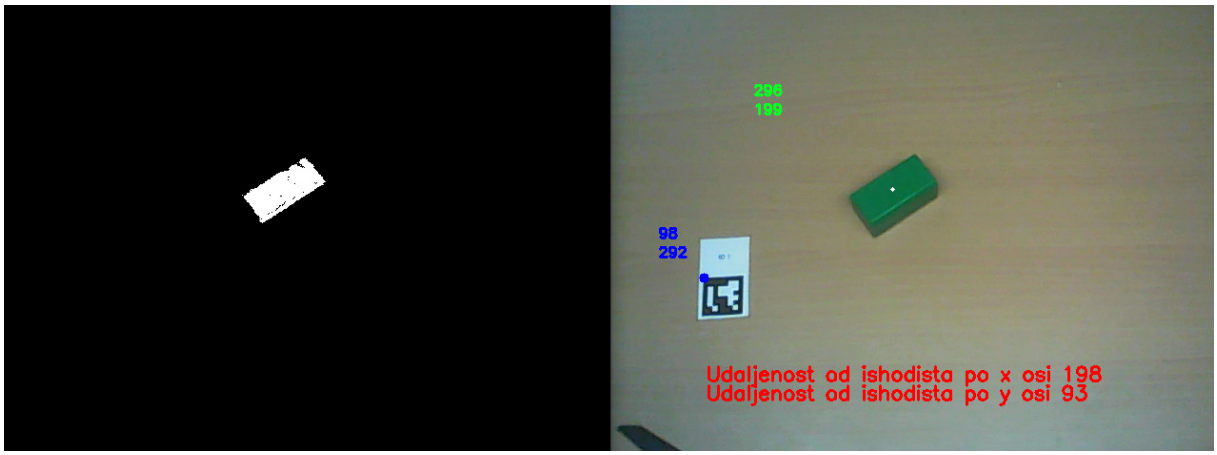
Slika 4.15. Pozicija ruba ArUco markera

Kao i kod detekcije položaja težišta, OpenCV kao rezultat položaja markera daje informaciju o horizontalnoj i vertikalnoj udaljenosti između gornjeg lijevog ruba slike i piksela koji predstavlja pronađeni rub markera. S obzirom da se oba položaja računaju kao udaljenosti od istog ishodišta vrlo jednostavno može se napisati matematički izraz pomoću kojega se računa relativna udaljenost između položaja markera i položaja težišta određenog objekta. Slika 4.16 prikazuje dio koda kojim se poziva traženi program.

```
import cv2
import numpy as np
from cv2 import aruco

cap = cv2.VideoCapture(0)

def findAruco(frame, marker_size=6, total_markers=250):
    gray=cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    key=getattr(aruco,f'DICT_{marker_size}X{marker_size}_{total_markers}')
    arucoDict=aruco.Dictionary_get(key)
    arucoParam=aruco.DetectorParameters_create()
    bbox,ids,_ =aruco.detectMarkers(gray,arucoDict,parameters=arucoParam)
    marker_corners, ids, _ = aruco.detectMarkers(gray, arucoDict, parameters=arucoParam)

    if marker_corners:
        x1 = int (marker_corners[0][0][0][0])
        y1 = int (marker_corners[0][0][0][1])
        cv2.putText(frame, str(x1), (50, 250), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)
        cv2.putText(frame, str(y1), (50, 270), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)
        cv2.circle(frame, (x1, y1), 5, (255,0,0), -1)
        cv2.putText(frame, "Udaljenost od ishodista po x osi " +str(cx-x1), (100, 400),
        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
        cv2.putText(frame, "Udaljenost od ishodista po y osi " +str(-(cy-y1)), (100, 420),
        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
    return ids, bbox

while True:
    _, frame = cap.read()
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    lower = np.array([47, 133, 107])
    upper = np.array([86, 255, 255])
    mask = cv2.inRange(hsv, lower, upper)

    cnts, hei = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    for c in cnts:
        area = cv2.contourArea(c)
        if area > 300:
            M=cv2.moments(c)
            cx= int(M["m10"]/M["m00"])
            cy= int(M["m01"]/M["m00"])
            cv2.circle(frame, (cx, cy), 2, (255,255,255), -1)
            cv2.putText(frame, str(cx), (150, 100), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (36, 255, 12), 2)
            cv2.putText(frame, str(cy), (150, 120), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (36, 255, 12), 2)

    bbox, ids = findAruco(frame)
    cv2.imshow("frame", frame)
    cv2.imshow("mask", mask)
    if cv2.waitKey(20) & 0xFF==ord('d'):
        break
cap.release()
cv2.destroyAllWindows()
```

Slika 4.16. Program za izračunavanje udaljenosti od ishodišta

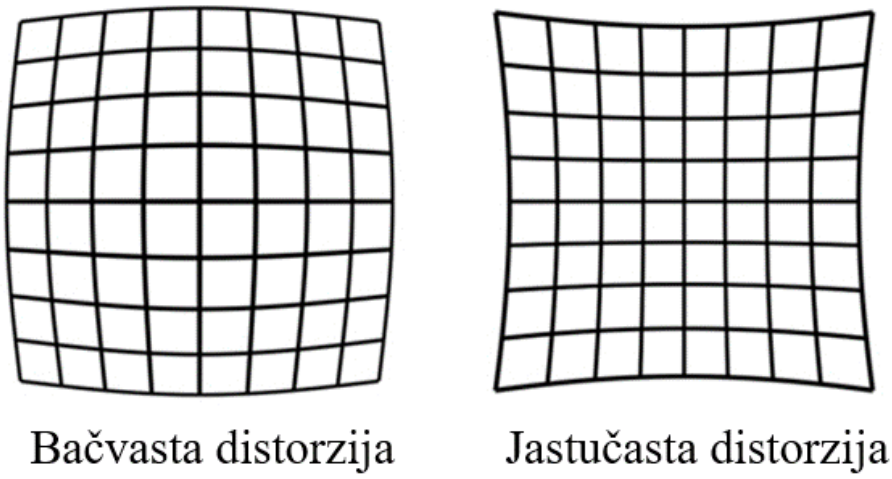
Slika 4.17 prikazuje krajnji rezultat koji daje udaljenost između ruba markera, koji se koristi kao ishodište i težišta predmeta koji se analizira. Plavim znamenkama iskazan je položaj ruba markera, dok je zelenim znamenkama naznačen položaj težišta zelenog kvadra. Crvenim slovima napisana je tražena vrijednost udaljenosti centra predmeta od ishodišta. Ponovno je vrijednost iskazana u pikselima i bit će potrebno provesti pretvorbu u milimetre u kasnijim koracima programa.



Slika 4.17. Udaljenost između ArUco markera i težišta predmeta

4.5. Kalibracija kamere

U okviru ovog rada kao vizijski senzor koji analizira radnu okolinu i daje sliku s koje se očitavaju koordinate koje se dalje šalju na robota koristi se web kamera. Kada se kamera koristi kao vizijski senzor nužno je poznavati parametre kamere kako bi se mogle ukloniti radijalne distorzije koje nastaju, a dijele se na dva oblika, jastučasti i bačvasti. Slika 4.18 prikazuje spomenute distorzije.



Slika 4.18. Radijalne distorzije [18]

Postupak izračuna parametara kamere naziva se kalibracija kamere. Tim postupkom dobivaju se sve ključne informacije u obliku parametara ili koeficijenata potrebne za određivanje preciznog odnosa između trodimenzionalne točke u stvarnom svijetu i odgovarajuće dvodimenzionalne projekcije te točke u obliku piksela na slici snimljenoj putem kamere. Obično se dobivaju dvije skupine parametara. Prva skupina odnosi se na unutarnje parametre kamere i leće, a obuhvaćaju žarišnu duljinu, optički centar i koeficijente radijalne distorzije leće. Druga skupina su vanjski parametri kamere, a oni se odnose na orijentaciju koja uključuje translaciju i rotaciju kamere u odnosu na određeni vanjski koordinatni sustav. [19] Slika 4.19 prikazuje upotrebu kalibracije kamere u svrhu uklanjanja radijalnih distorzija na slici.

Slika 4.19. Uklanjanje radijalnih distorzija [20]

Kako bi se pronašla projekcija 3D točke u ravnini slike najprije je potrebno izvršiti transformaciju točke iz vanjskog koordinatnog sustava, onog u stvarnom svijetu, u koordinatni sustav kamere koristeći ekstrinzične parametre (rotacija i translacija). Idući korak je korištenjem intrinzičnih parametara kamere projicirati točku na ravninu slike. Jednadžbe (5), (6) i (7) koje povezuju 3D točku (X_w, Y_w, Z_w) u vanjskim koordinatama s njezinom projekcijom (u, v) u koordinatama slike prikazane su u nastavku: [19]

$$\begin{bmatrix} u' \\ v' \\ z' \end{bmatrix} = \mathbf{P} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{5}$$

$$u = \frac{u'}{w'} \tag{6}$$

$$v = \frac{v'}{w'} \tag{7}$$

Pritom je **P** 3×4 projekcijska matrica koja se sastoji od intrinzične matrice **K** koja sadrži intrinzične parametre i ekstrinzičnu matricu [**R**|**t**] koja je kombinacija 3×3 matrice **R** i 3×1 translacijskog vektora **t** što je prikazano jednadžbom (8).

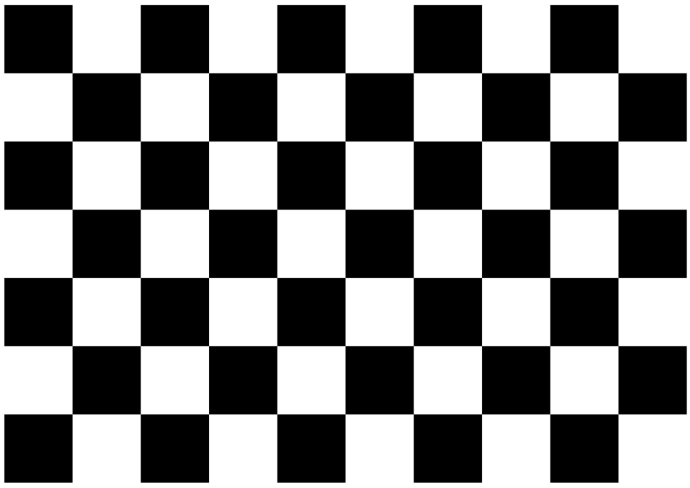
$$\mathbf{P} = \mathbf{K} \times \left[\mathbf{R} \mid \mathbf{t} \right] \tag{8}$$

Intrinzična matrica **K** je gornja trokutasta matrica oblika:

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

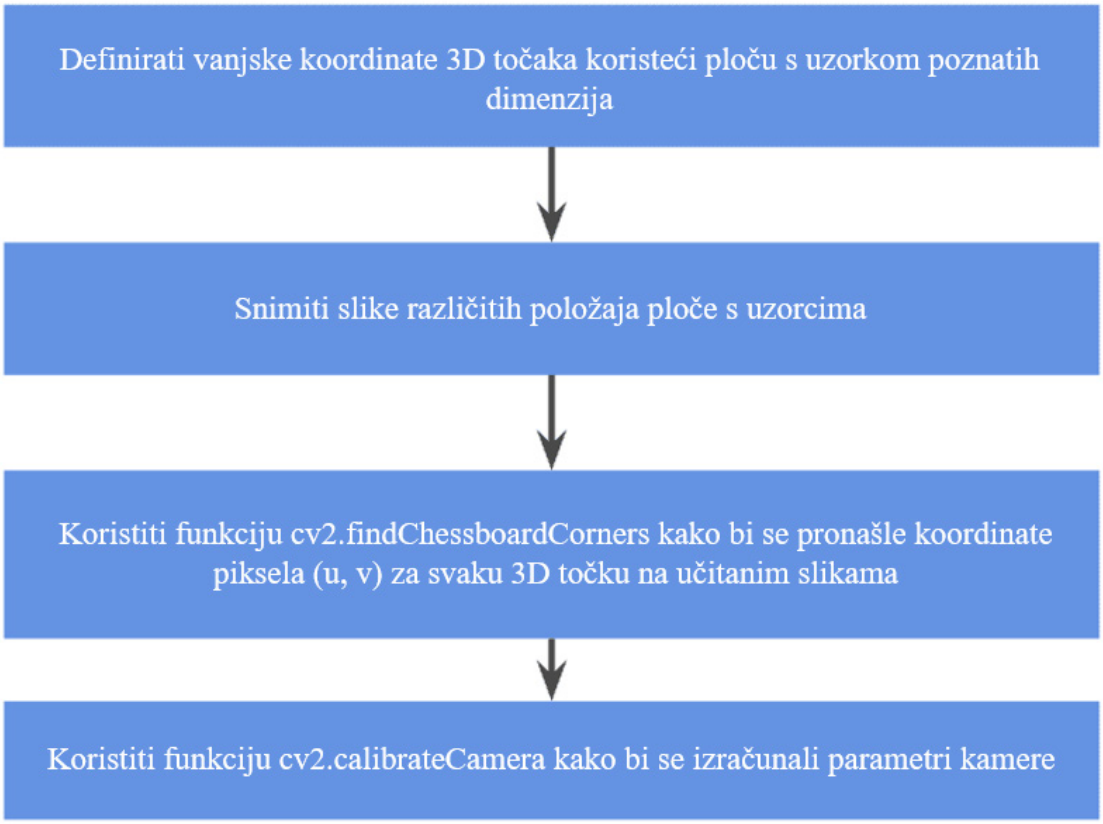
Gdje su f_x i f_y žarišne duljine koje su obično jednake, dok su c_x i c_y su koordinate optičkog centra u ravnini slike. Cilj postupka kalibracije je pronalaženje 3×3 matrice **K**, rotacijske matrice **R** i translacijskog vektora **t** korištenjem niza poznatih 3D točaka (X_w, Y_w, Z_w) i njihovih koordinata na slici (u, v). Kamera je kalibrirana onda kada su izračunate vrijednosti intrinzičnih i ekstrinzičnih parametara. Postoji nekoliko metoda kojima se provodi kalibracija kamere, a među njih spada metoda korištenja kalibracijskog uzorka. Uzorci mogu biti raznih oblika, ali onaj najčešći koji je ujedno i korišten pri kalibraciji u sklopu ovog rada je uzorak „šahovske“ ploče crno-bijelih kvadrata. [19]

Slika 4.20 prikazuje primjer takvog uzorka.



Slika 4.20. Uzorak za kalibraciju [20]

Slika 4.21 prikazuje dijagram toka na kojem je opisan postupak kalibracije unutar OpenCV-a.



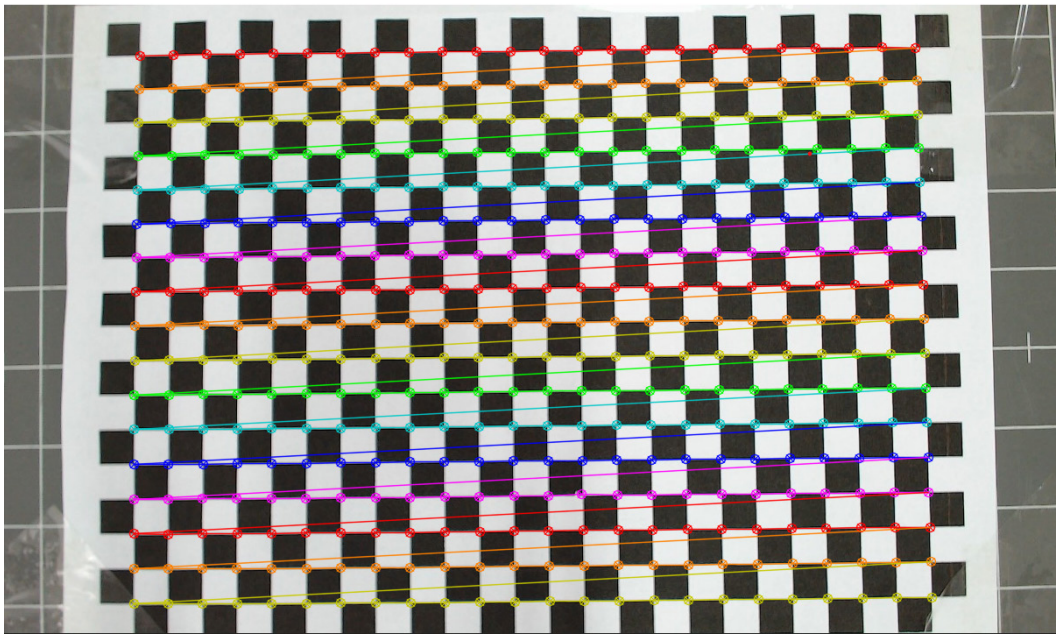
Slika 4.21. Dijagram toka kalibracije kamere

Kao što je na dijagramu naznačeno u prvom koraku potrebno je definirati vanjske koordinate 3D točaka pomoću uzorka poznatog oblika. Nakon što su slike snimljene i učitane, koristi se funkcija cv2.findChessboardCorners kako bi se pronašle koordinate piksela (u, v) za svaku 3D točku. U zadnjem koraku izračunavaju se parametri kamere korištenjem funkcije cv2.calibrateCamera. Slika 4.22 prikazuje dio koda koji se koristi za dobivanje parametara potrebnih za kalibraciju kamere.

```python
import numpy as np
import cv2 as cv
import glob

chessboardSize = (24,17)
frameSize = (1280,720)

criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)

objp = np.zeros((chessboardSize[0] * chessboardSize[1], 3), np.float32)
objp[:,:2] = np.mgrid[0:chessboardSize[0],0:chessboardSize[1]].T.reshape(-1,2)

objpoints = [] # 3d point in real world space
imgpoints = [] # 2d points in image plane

images = glob.glob('kalibracija/*.jpg')

for image in images:
    img = cv.imread(image)
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

    ret, corners = cv.findChessboardCorners(gray, chessboardSize, None)
    if ret == True:

        objpoints.append(objp)
        corners2 = cv.cornerSubPix(gray, corners, (11,11), (-1,-1), criteria)
        imgpoints.append(corners)
        cv.drawChessboardCorners(img, chessboardSize, corners2, ret)
        cv.imshow('img', img)
        cv.waitKey(1000)

cv.destroyAllWindows()

ret, cameraMatrix, dist, rvecs, tvecs = cv.calibrateCamera(objpoints, imgpoints, frameSize, None, None)

print("Camera calibrated: ", ret)
print("\nCamera Matrix:\n", cameraMatrix)
print("\nDistortion Parameters:\n", dist)
print("\nRotation vectors:\n", rvecs)
print("\nTranslation vectors:\n", tvecs)
```

Slika 4.22. Program za kalibraciju kamere

Slika 4.23 prikazuje pronađene točke na učitanoj slici korištenoj za kalibraciju. U ovom slučaju korištena je ploča s 24 horizontalna ruba i 17 vertikalnih rubova.



Slika 4.23. Pronađene točke na slici za kalibraciju

Slika 4.24 prikazuje izlazne vrijednosti programa koje između ostalog sadrže matricu kalibracije i koeficijente distorzije koji su potrebni za uklanjanje distorzije kamere.

```
Camera Matrix:
 [[2.26763495e+03 0.00000000e+00 6.40692236e+02]
 [0.00000000e+00 2.28577970e+03 3.68056793e+02]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]

Distortion Parameters:
 [[ 2.18046482e-01 -4.75871871e+00 -2.39458777e-03  2.47083469e-03
   1.42506296e+01]]

Rotation vectors:
 (array([[-0.10594953],
       [-0.00283823],
       [-0.00980534]]),)

Translation vectors:
 (array([[-12.17150177],
       [ -7.99091507],
       [ 58.84809964]]),)
```

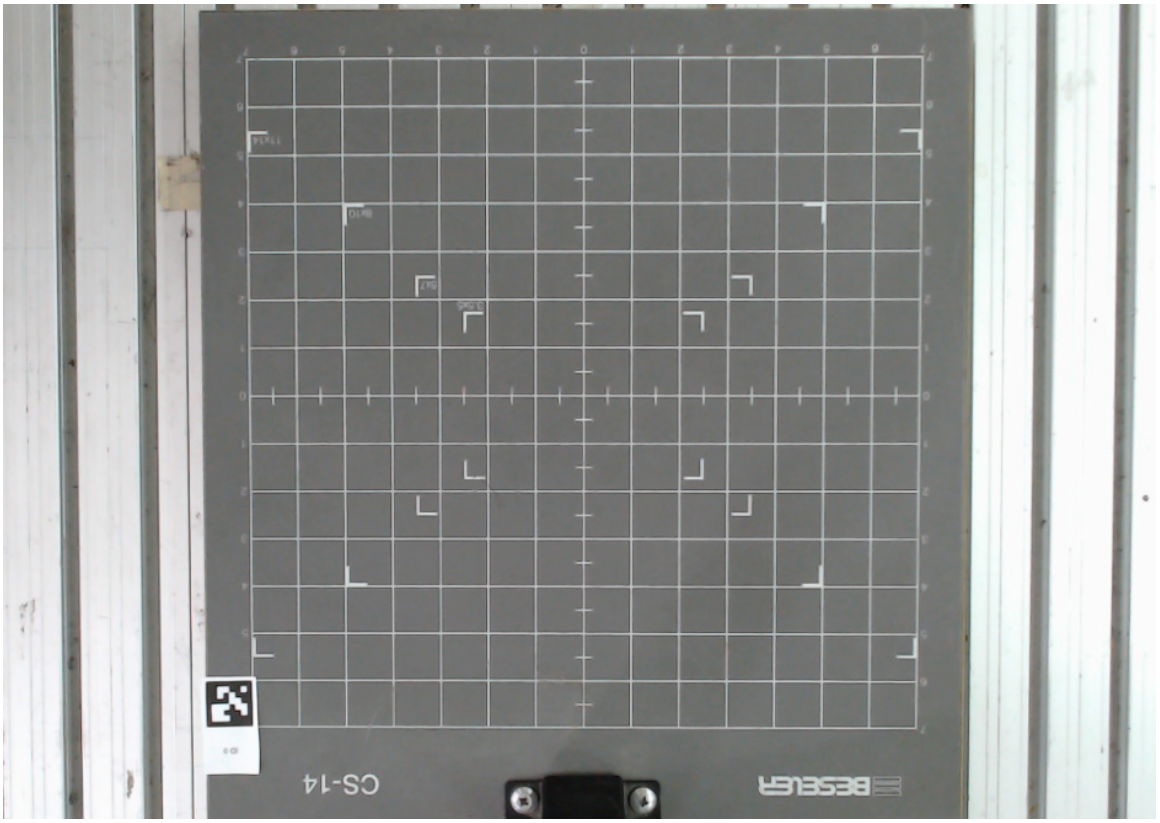
Slika 4.24. Izlazne vrijednosti kalibracije kamere

Nakon što su izračunati svi parametri koristi se funkcija cv2.getOptimalNewCameraMatrix kako bi se optimirali rezultati dobiveni kalibracijom nakon čega se koriste novo izračunati parametri za uklanjanje distorzije sa slike. Slika 4.25 prikazuje linije koda korištene za uklanjanje distorzija.

```
cap = cv.VideoCapture(1)
cap.set(cv.CAP_PROP_FRAME_WIDTH, 1280)
cap.set(cv.CAP_PROP_FRAME_HEIGHT, 720)

while True:
    isTrue, frame = cap.read()
    h,  w = frame.shape[:2]
    newCameraMatrix, roi = cv.getOptimalNewCameraMatrix(cameraMatrix, dist, (w,h), 1, (w,h))
    dst = cv.undistort(frame, cameraMatrix, dist, None, newCameraMatrix)
```

Slika 4.25. Program za optimiranje rezultata kalibracije

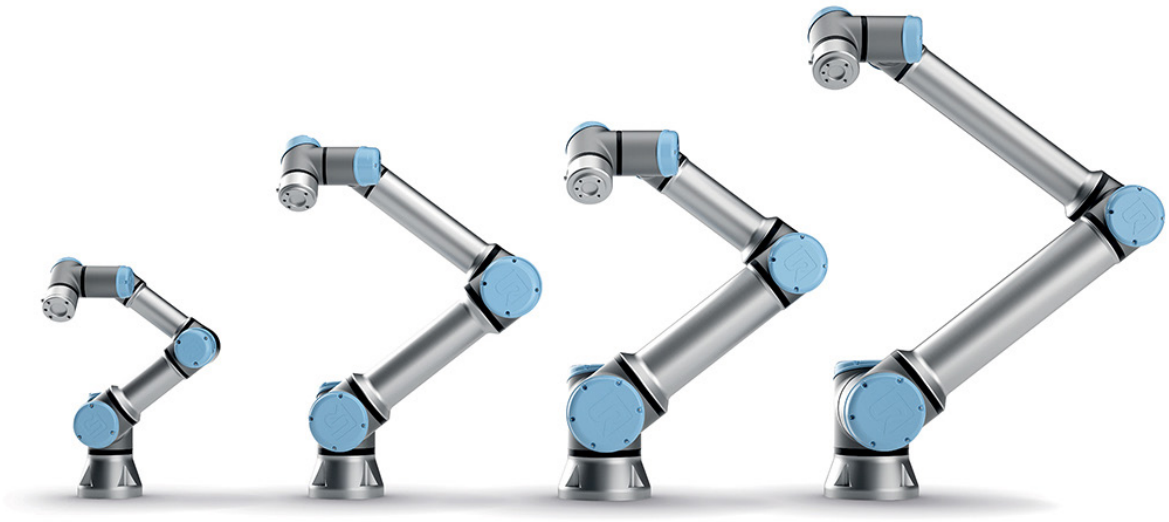
Slika 4.26 prikazuje konačni izgled radnog prostora nakon kalibracije. Može se vidjeti da su linije kvadratnih polja na radnoj ploči ravne i međusobne okomite što omogućava izračun položaja predmeta smještenog na ploči i slanje izračunatih koordinata na robota kako bi se izvršilo izuzimanje predmeta.



Slika 4.26. Radni prostor nakon kalibracije kamere

5. ODABRANI ROBOTSKI SUSTAV

U sklopu ovog rada potrebno je odabrati robotski sustav koji je pristupačan i jednostavan za korištenje u ljudskom okruženju. S obzirom da su predmeti rada u sklopu ovog zadatka malih dimenzija i masa faktor nosivosti koji robot treba ispuniti malog je iznosa i stoga je donesena odluka da se za realizaciju ovog zadatka koristi robot tvrtke Universal Robots. Riječ je o danskom proizvođaču malih i fleksibilnih kolaborativnih robota. Tvrtka je nastala nakon što su osnivači tvrtke Esben Østergaard, Kasper Støy i Kristian Kassow proveli zajedničko istraživanje na Sveučilištu Syddansk i došli do zaključka da na tržištu robota dominiraju skupi, teški i nezgrapni roboti. Po njihovom mišljenju robotske tehnologije trebale bi biti dostupne malim i srednjim poduzećima i vodeći se tom idejom na tržište je 2009. godine plasiran prvi robot tvrtke Universal Robots naziva UR5, robot koji će biti korišten u sklopu ovog rada. U međuvremenu je tvrtka proizvela modele UR3, UR10 i UR16 koji su zajedno s modelom UR5 prikazani na slici (Slika 5.1). [21]



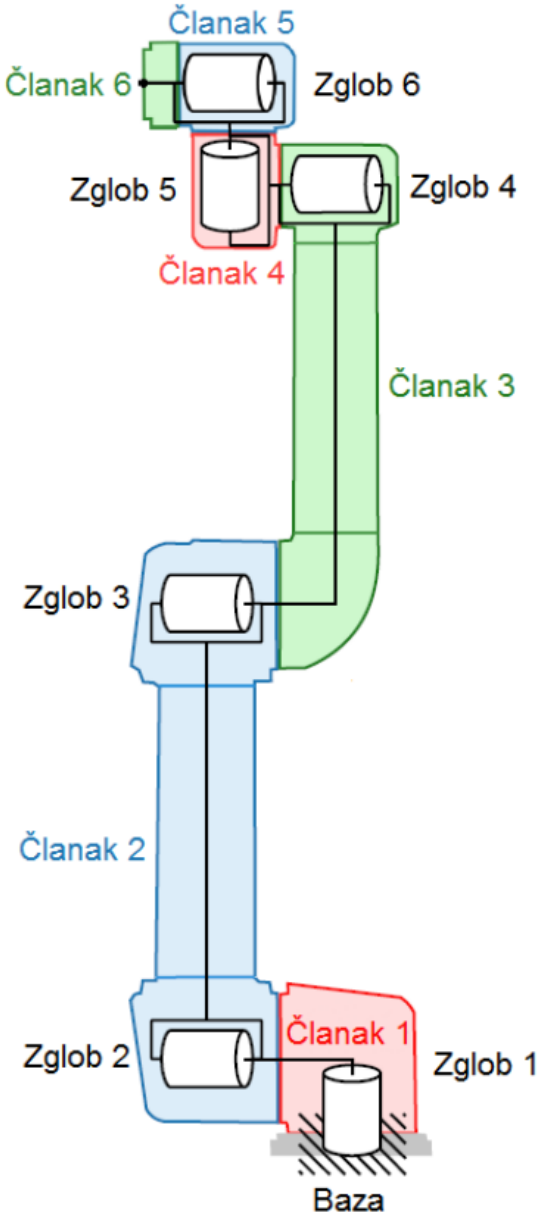
Slika 5.1. UR3, UR5, UR10 i UR16 [22]

Brojevi pokraj imena označavaju nosivost u kilogramima. Treba naglasiti da svi ovi roboti spadaju u skupinu kolaborativnih robota, što znači da su pogodni za rad u ljudskom okruženju, bez potrebe za sigurnosnim kavezima. Sva četiri modela su šesteroosna, s brzinom rotacije zglobova od 180 °/s uz točnost ponavljanja od ± 0,1 mm. Tablica 5.1 prikazuje detaljne karakteristike robota UR5 koji je korišten u sklopu ovog rada.

Tablica 5.1. Tehničke karakteristike robota UR5

UR5	
Masa:	18,4 kg
Nosivost:	5 kg
Doseg:	850 mm
Radni opseg zglobova:	± 360°
Brzina zglobova:	180 °/s
Brzina alata:	1 mm/s
Ponovljivost:	± 0,1 mm
Broj stupnjeva slobode gibanja:	6
Komunikacija:	TCP/IP 100 Mbit, Modbus TCP
Temperaturno radno područje:	0 – 50 °C
Materijali:	Aluminij i Polipropilen (PP)
Napajanje:	100 – 240V AC, 50 – 60 Hz
Potrošnja energije:	~ 200 W

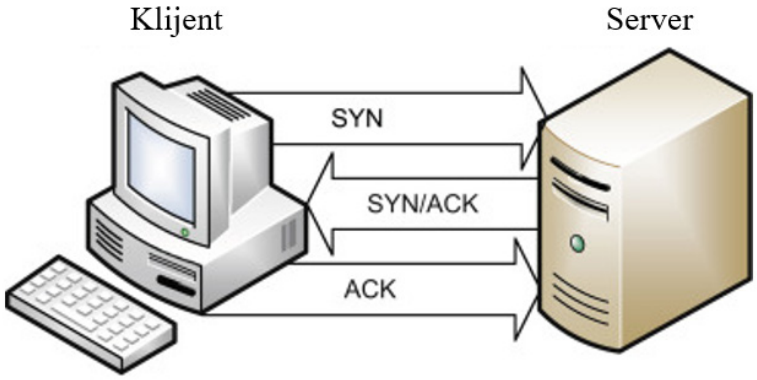
Slika 5.2 prikazuje sve zglobove i članke navedenog robota.



Slika 5.2. Zglobovi i članci UR5 robota [23]

Komunikacija između robota i računala vrši se preko TCP protokola (eng. *Transmission Control Protocol*), jednog od glavnih protokola unutar IP (eng. *Internet Protocol*) skupine. TCP pruža pouzdanu i uređenu isporuku niza bajtova koja je prethodno provjerena na pogreške. Mnoge usluge na Internetu kao što su WWW (eng. *World Wide Web*), e-pošta, upravljanje na daljinu i prenošenje podataka koriste TCP protokol koji je dio transportnog sloja TCP/IP infrastrukture. TCP je orijentiran na povezanost, a veza između klijenta i servera mora biti uspostavljena prije nego što podaci mogu biti poslani. Korisnici se mogu nalaziti u ulozi servera ili u ulozi klijenta, a unutar veze postoji jedan server i jedan ili više klijenata. Nakon što se

pokrene server veza se uspostavlja na način da klijenti koji se žele povezati šalju SYN (eng. *synchronize*) segment, dok server za prihvaćanje odgovara sa SYN-ACK (eng. *synchronize-acknowledge*) segmentom. Klijent potom šalju ACK (eng. *acknowledge*) segment čime se potvrđuje povezivanje. Opisana procedura naziva se trostruko rukovanje i može se vidjeti na slici (Slika 5.3), a kao uvjet za provedbu nužno je da se klijent i server nalaze unutar iste podmreže. [24]



Slika 5.3. Trostruko rukovanje TCP protokola [25]

U slučaju ovog rada računalo se nalazi u ulozi servera, dok je robot u ulozi klijenta. Sama komunikacija izvršava se preko Python knjižnice naziva urx pomoću koje je moguće kontrolirati robota. Nakon što su računalo i robot spojeni na istu lokalnu mrežu definira se varijabla robota pozivom funkcije urx.robot u koju se unosi IP adresa dodijeljena robotu. Nakon toga moguće je putem pozivanja niza unaprijed definiranih funkcija ostvariti pomak robotskih zglobova.

6. IZRADA ROBOTSKE HVATALJKE

Idući korak u sklopu ovog rada je izrada adekvatne robotske hvataljke upotrebom aditivnih tehnologija. Kod izrade hvataljke bitno je analizirati geometriju predmeta koji se koriste kao radni komadi kod robotskog izuzimanja. Na osnovu njihovih dimenzija i oblika potrebno je oblikovati hvataljku koja se uspješno može koristiti za rad s robotom uzimajući u obzir moguću grešku pozicioniranja koju je preko oblika hvataljke moguće anulirati.

6.1. Aditivna proizvodnja

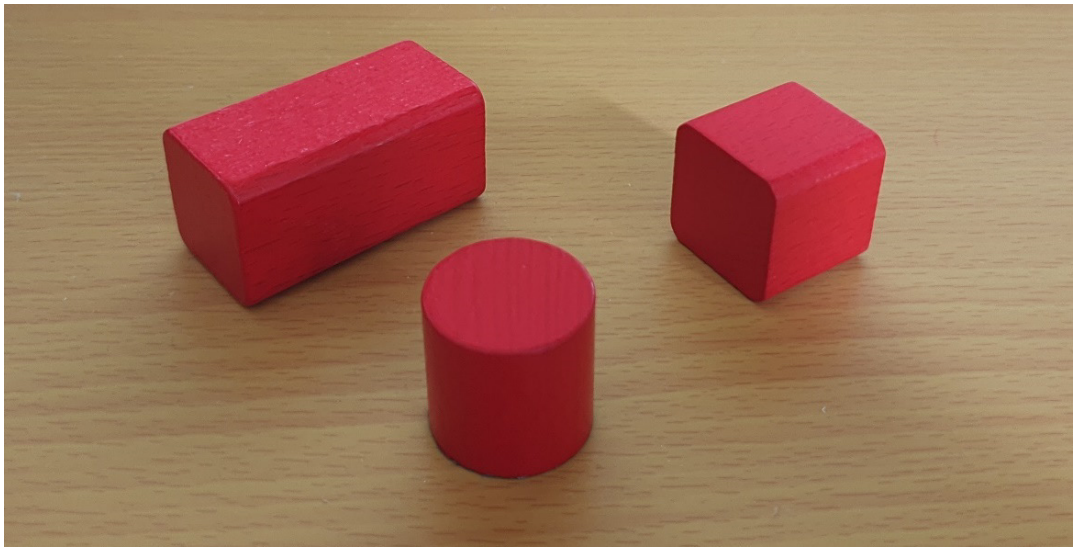
Aditivna proizvodnja ili 3D ispisivanje je izgradnja trodimenzionalnog objekta iz CAD (eng. *Computer Aided Design*) modela ili digitalnog 3D modela. Tehnologija je razvijena 80-ih godina prošlog stoljeća i u to vrijeme koristila se samo za izradu prototipova. U međuvremenu su se karakteristike postupka kao što su preciznost, ponovljivost i raspon korištenih materijala povećali do te mjere da se danas postupci 3D ispisivanja koriste kao sastavna tehnologija industrijske proizvodnje gotovih proizvoda. Osnovni princip aditivne proizvodnje je dodavanje materijala sloj po sloj pri čemu nema gotovo nikakvih gubitaka materijala što je prednost u odnosu na konvencionalne postupke proizvodnje koji se temelje na skidanju materijala s obratka pri čemu se sav odstranjeni materijal smatra gubitkom. Još jedna prednost aditivne proizvodnje je mogućnost izrade predmeta vrlo složenih oblika i geometrije koje je nemoguće proizvesti klasičnim postupcima obrade jer sadržavaju razne šupljine preko kojih se ostvaruje smanjena masa predmeta. FDM (eng. *Fused deposition modeling*) je metoda 3D ispisivanja koja koristi kontinuirani filament termoplastičnog materijala i ujedno je najčešća metoda 3D ispisa te će se koristiti pri izradi predmeta u ovom radu. [26]

6.2. Oblikovanje hvataljke

Prvi korak pri oblikovanju hvataljke je izrada CAD modela potrebnog za postupak 3D ispisa. Prilikom modeliranja ključno je proučiti geometriju predmeta rada koji će biti korišteni za rad s hvataljkom. Na osnovu njihovih oblika i dimenzija oblikuje se hvataljka unutar određenog CAD alata, a za potrebe ovog rada odabran je programski paket Inventor. Slika 6.1 prikazuje predmete rada, dok su u tablici (Tablica 6.1) dane njihove gabaritne dimenzije s kojima je hvataljka u direktnom dodiru.

Tablica 6.1. Dimenzije predmeta rada

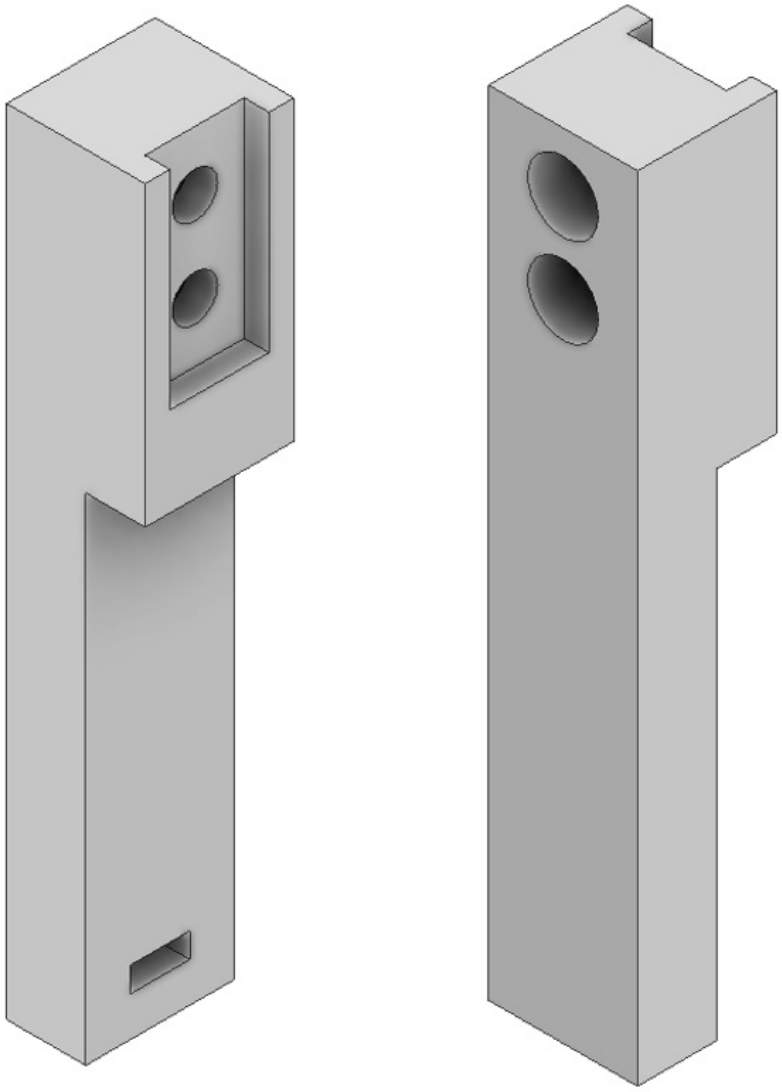
Naziv predmeta	Kvadar	Kocka	Cilindar
Dimenzije baze (mm)	50×25	25×25	ϕ25
Visina (mm)	25	25	25



Slika 6.1. Predmeti rada

S obzirom da je potrebno napraviti univerzalnu hvataljku s kojom je moguće izvršiti izuzimanje sve tri vrste predmeta prilikom modeliranja hvataljke potrebno je osmisliti geometriju koja će ostvariti kontakt s predmetima rada u više od dvije točke radi što čvršćeg i stabilnijeg prihvata. Hvataljka je izrađena iz dva dijela, a to su prsti hvataljke i nastavak hvataljke. Prsti hvataljke odnose se na element koji se povezuje na pneumatsku hvataljku koja je pričvršćena direktno na robotsku prirubnicu, dok je nastavak hvataljke element koji je u direktnom kontaktu s predmetima rada kada je hvataljka u zatvorenom položaju.

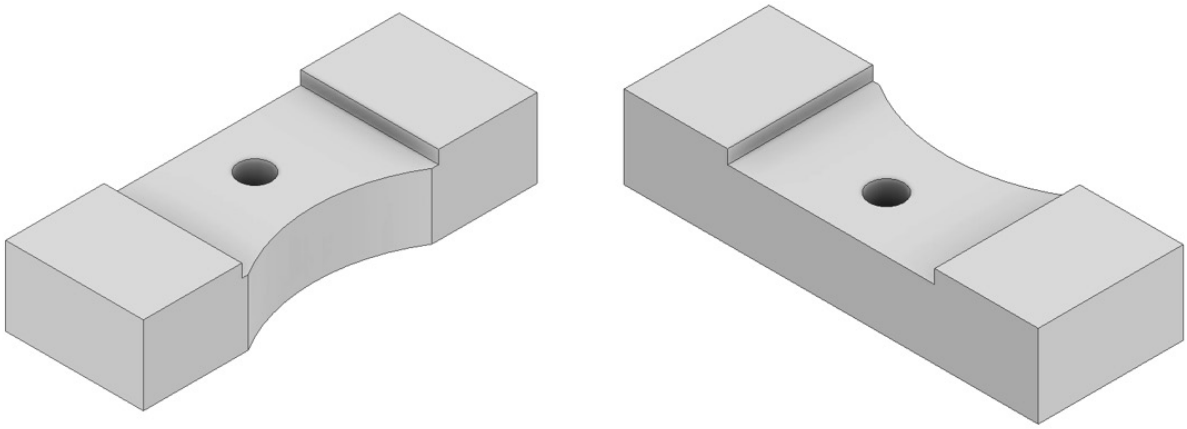
Slika 6.2 prikazuje prste za hvataljku koji s donje strane imaju provrt kroz koji prolazi vijak M 2,5 koji preko matice koja ulazi kroz šupljinu s bočne strane predmeta ostvaruje čvrsti spoj s nastavkom hvataljke.



Slika 6.2. Prsti hvataljke

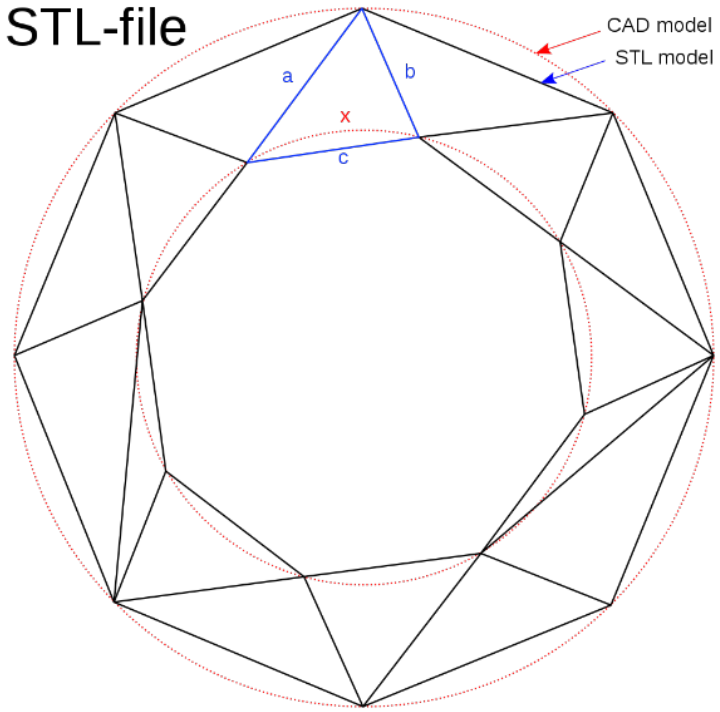
Zamišljeno je da se nastavak hvataljke mijenja u ovisnosti o zahtjevima predmeta rada. U prilogu na kraju rada dana je tehnička dokumentacija za oblikovane predmete.

Nastavak hvataljke oblikovan za potrebe ovog zadatka prikazan je na slici (Slika 6.3).



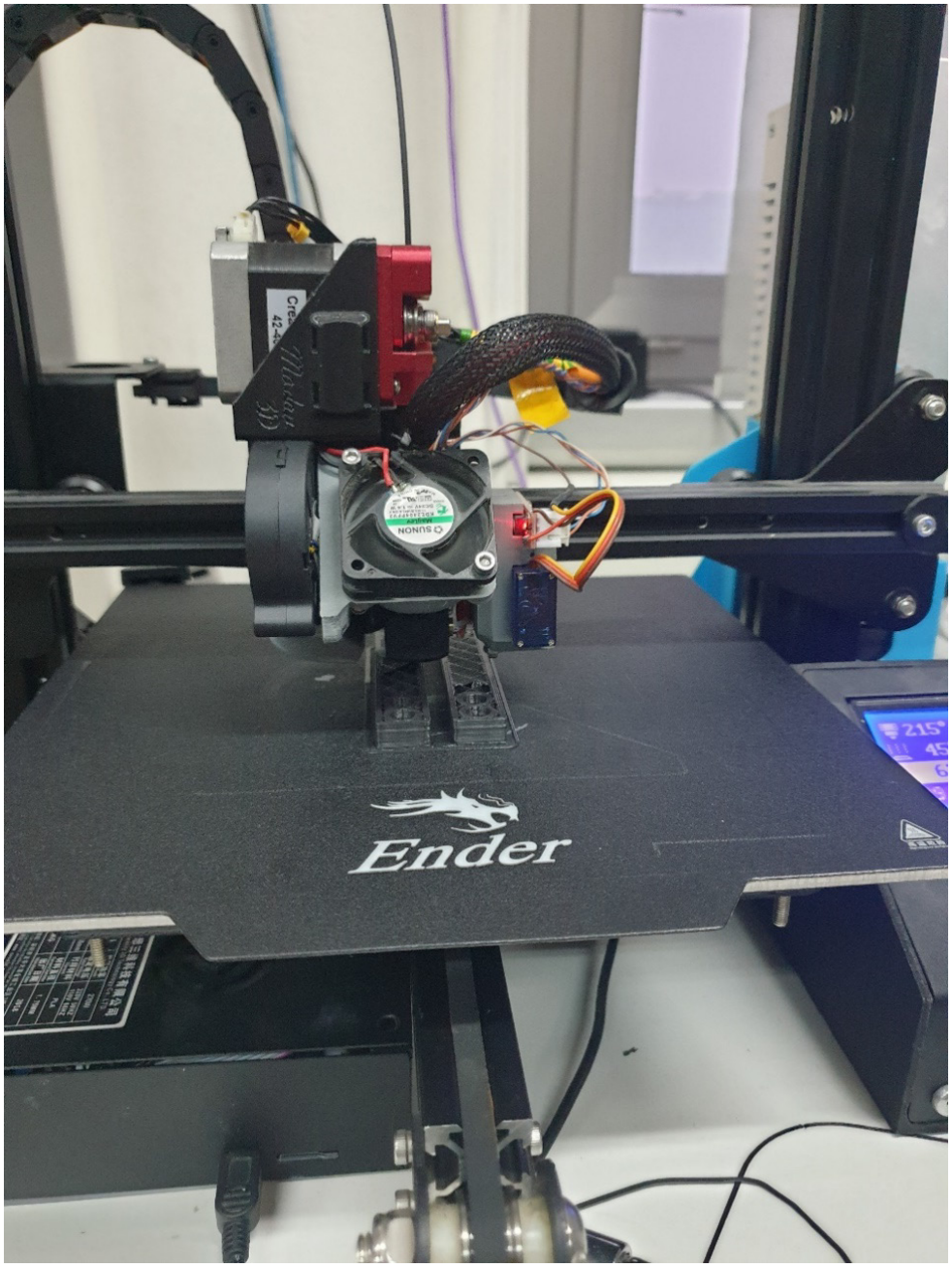
Slika 6.3. Nastavak hvataljke

Nakon što su modeli oblikovani pomoću CAD programskog paketa Inventor potrebno je izvršiti pretvorbu datoteke u STL (eng. *stereolithography*) format kojeg podržava većina programskih paketa koji se koriste za 3D oblikovanje. STL format prikazuje modele koristeći triangulaciju površine CAD modela. Razlika između prikaza kružnog prstena kod CAD i STL modela prikazana je na slici (Slika 6.4).



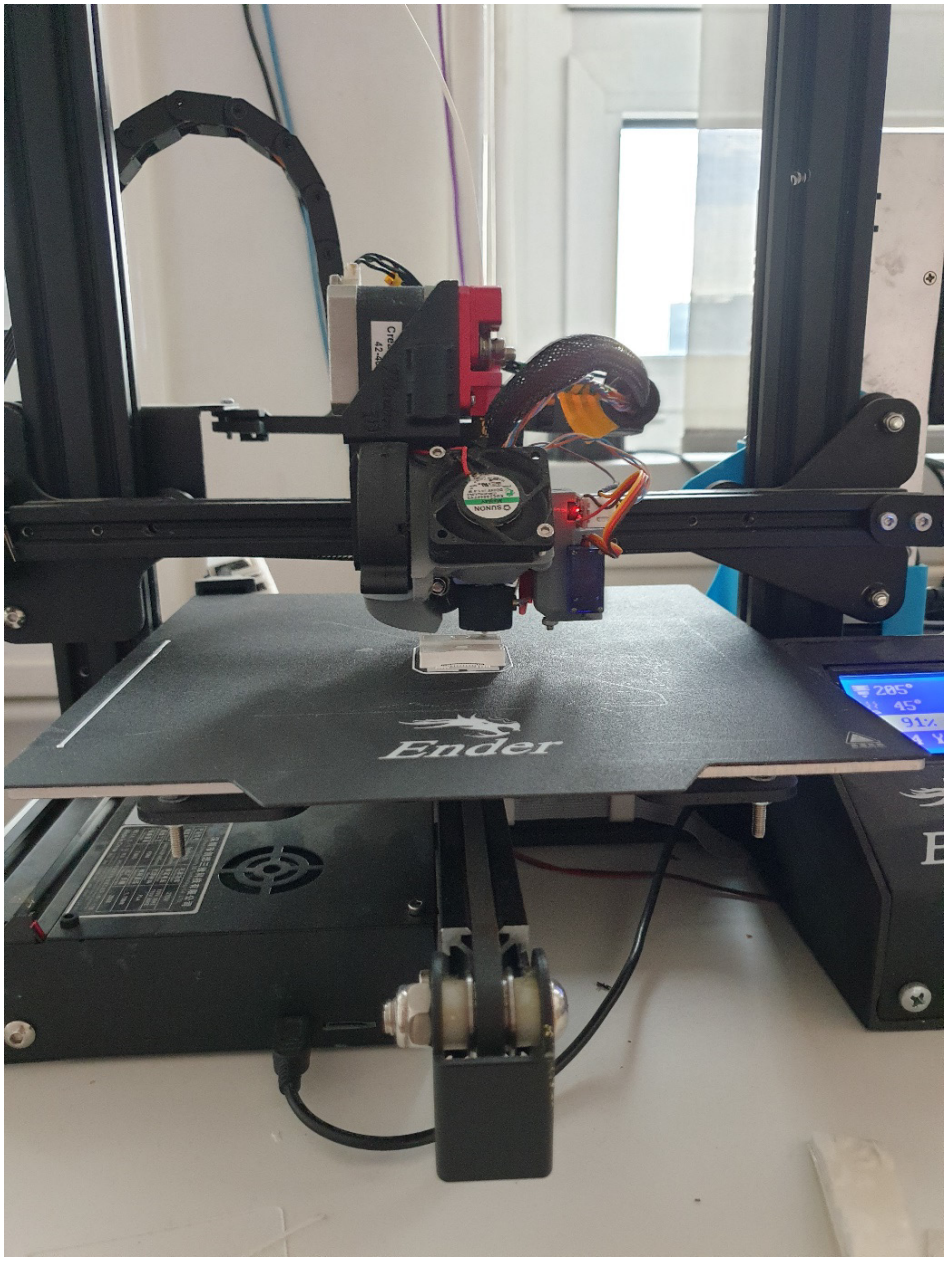
Slika 6.4. STL model [27]

Nakon što je STL datoteka provjerena na pogreške idući korak je generiranje G koda pomoću alata koji pripada skupini takozvanih slicer programa koji vrše pretvorbu 3D modela u niz tankih slojeva i generiraju G kod. G kod je skup naredbi koje opremi zadaju brzinu kretanja, položaje koje je potrebno zauzeti te upute o pravovremenom i ispravnom korištenju alata. Program naziva PrusaSlicer korišten je pri obradi STL datoteke u svrhu dobivanja G koda. Za ispisivanje elemenata hvataljke korišten je fakultetski 3D pisač. Materijal korišten za ispisivanje predmeta je PLA. Slika 6.5 prikazuje postupak ispisa prstiju hvataljke, dok Slika 6.6 prikazuje ispis nastavka hvataljke.



Slika 6.5. Ispis prstiju hvataljke

G kod šalje pisaču informacije potrebne za upravljanje elektromotorima, ekstruderom i podlogom koju je potrebno grijati. Grijač koji je smješten unutar ekstrudera topi korišteni materijal, dok ekstruder prima informacije o tome kojom brzinom te u kojem trenutku treba dodavati taljevinu. Pisač takoreći vrši pretvorbu trodimenzionalnog modela u niz poprečnih presjeka. Slojevi se slažu tako što se najprije ekstrudira jedan sloj te nakon što se taj sloj ohladi slijedi ekstruzija idućeg sloja. Postupak se ponavlja sve do potpunog ispisa predmeta.



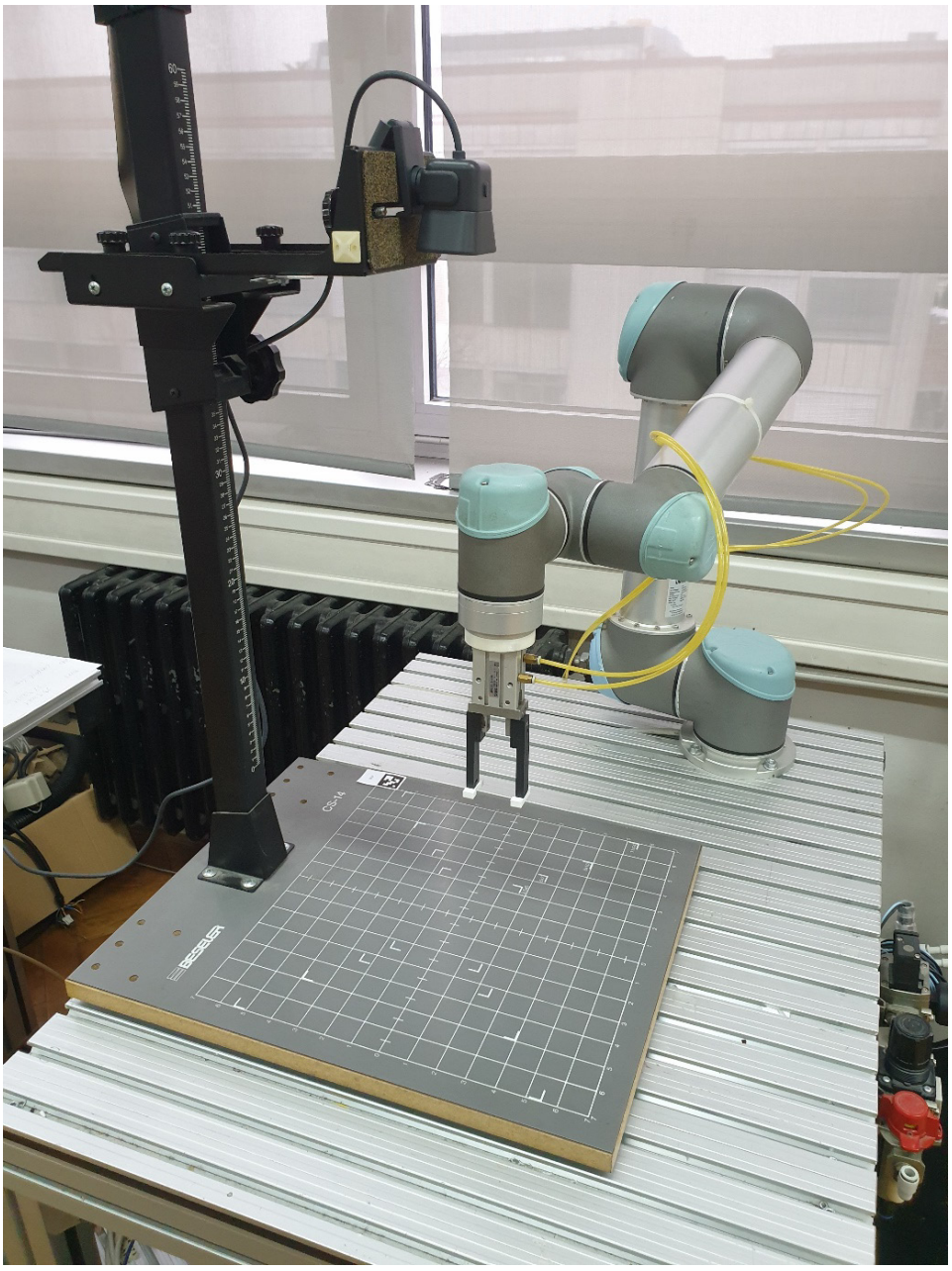
Slika 6.6. Ispis nastavka hvataljke

Nakon što su svi elementi ispisani preostalo je pričvrstiti hvataljku na robota. Konačni prikaz hvataljke spojene na robotsku prirubnicu može se vidjeti na slici (Slika 6.7).

Slika 6.7. Hvataljka spojena na robotsku prirubnicu

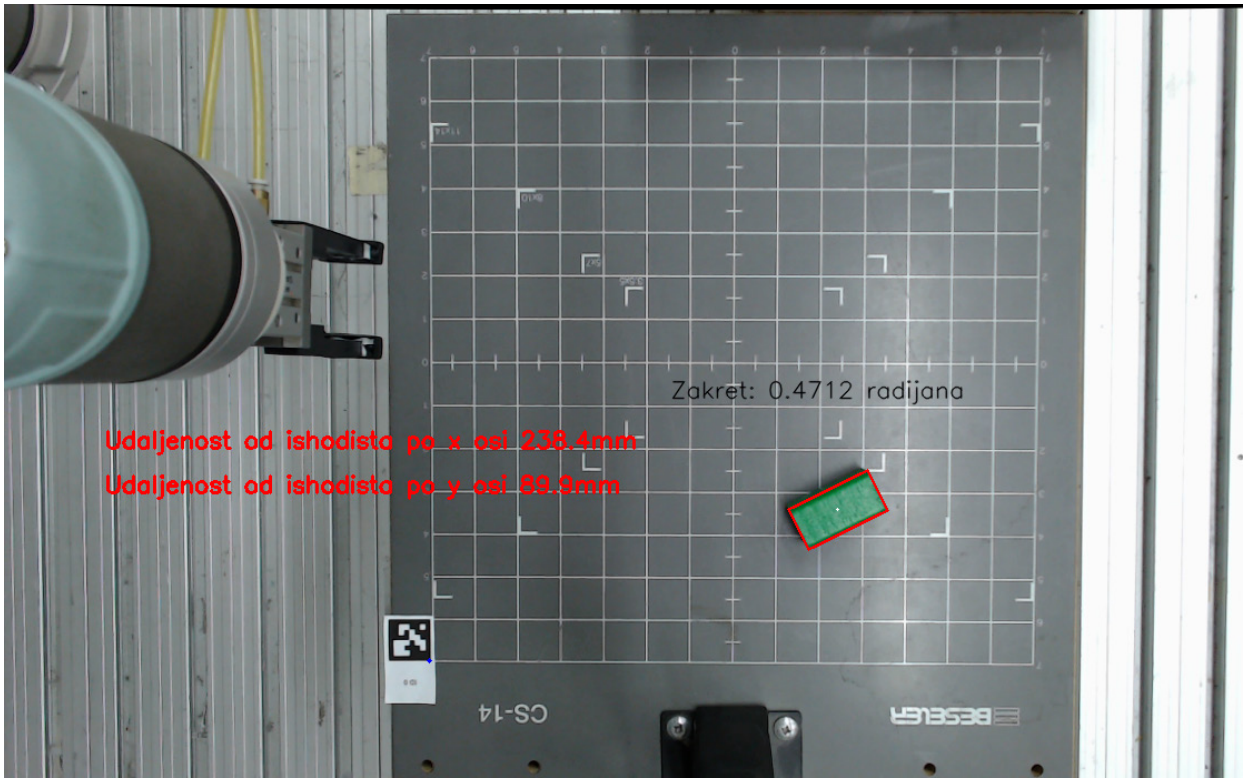
7. REZULTATI

Nakon što je programski kod napisan potrebno je izvršiti evaluaciju rezultata u stvarnoj okolini. Odabrano radno okruženje sastoji se od robotskog sustava UR5 na kojeg je spojena oblikovana hvataljka, dok je kamera postavljena na nosač koji je namješten na odgovarajuću visinu. Na dnu nosača nalazi se kvadratna ploča korištena za određivanje položaja radnih komada. Sve zajedno smješteno je na radni stol napravljen od aluminijskih profila. Slika 7.1 prikazuje radni postav u laboratoriju.



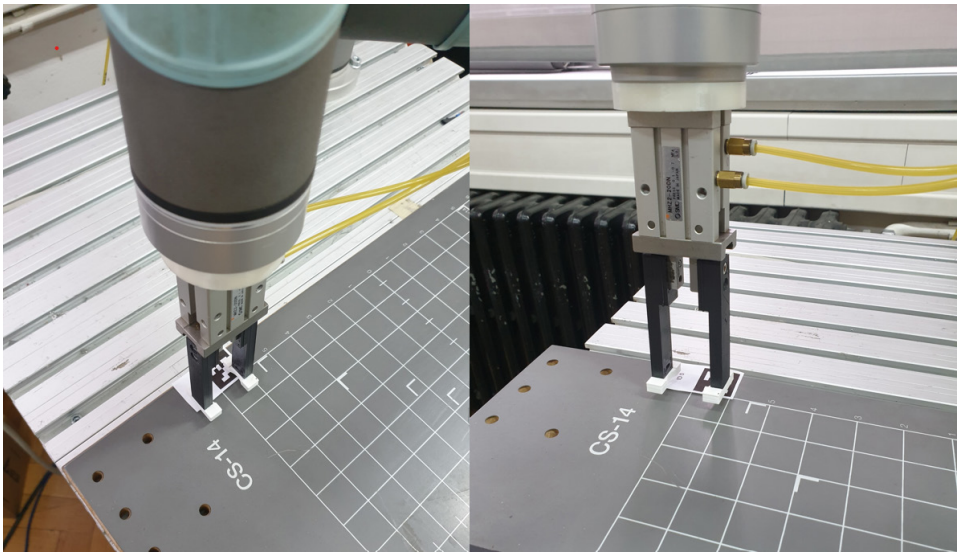
Slika 7.1. Radni postav

Pokretanjem programa za određivanje položaja i kutnog zakreta predmeta izračunata je udaljenost težišta predmeta s obzirom na ishodište za koje je odabran donji desni rub ArUco markera. Prethodno je izvršena pretvorba piksela u milimetre za zadanu visinu na kojoj je kamera postavljena na način da je broj piksela koji predstavlja određenu udaljenost podijeljen s brojem milimetara koji je izmjeren za traženu udaljenost. Slika 7.2 prikazuje izlaznu vrijednost programa.



Slika 7.2. Određivanje položaja i orijentacije predmeta u radnoj okolini

Idući korak je određivanje koordinate odabranog ishodišta u koordinatnom sustavu robota. To je odrađeno na način da se vrh alata robota dovede u poziciju iznad odabranog ishodišta, tj. ArUco markera koristeći robotski privjesak za učenje (eng. *teach pendant*) što je prikazano na slici (Slika 7.3).



Slika 7.3. Pozicioniranje hvataljke iznad ishodišta

Zatim se korištenjem urx knjižnice definiraju robotski parametri kao što su pozicija vrha alata (eng. *tool center point*) i nosivost (eng. *payload*) korištenjem funkcija prikazanih na slici (Slika 7.4).

```
import sys
import urx
import logging

def wait():
    if do_wait:
        print("Click enter to continue")
        input()

if __name__ == "__main__":
    logging.basicConfig(level=logging.INFO)

    do_wait = True
    if len(sys.argv) > 1:
        do_wait = False

    rob = urx.Robot ("192.168.0.25")
    rob.set_tcp((0, 0, 0.167, 0, 0, 0))
    rob.set_payload(0.5, (0, 0, 0))
```

Slika 7.4. Program za upravljanje robotom

Nakon toga se putem naredbe rob.getl prikazane na slici (Slika 7.5) dobiva trenutni položaj vrha alata prikazan u koordinatnom sustavu baze robota i te su koordinate spremljene u varijablu naziva pozicija.

```
rob.getl()
✓ 0.4s
[-0.2855348234598083,
 0.3870873627436223,
 0.10985646235193552,
 -2.2251806654567794,
 2.210201344281277,
 -0.017913692022315884]
```

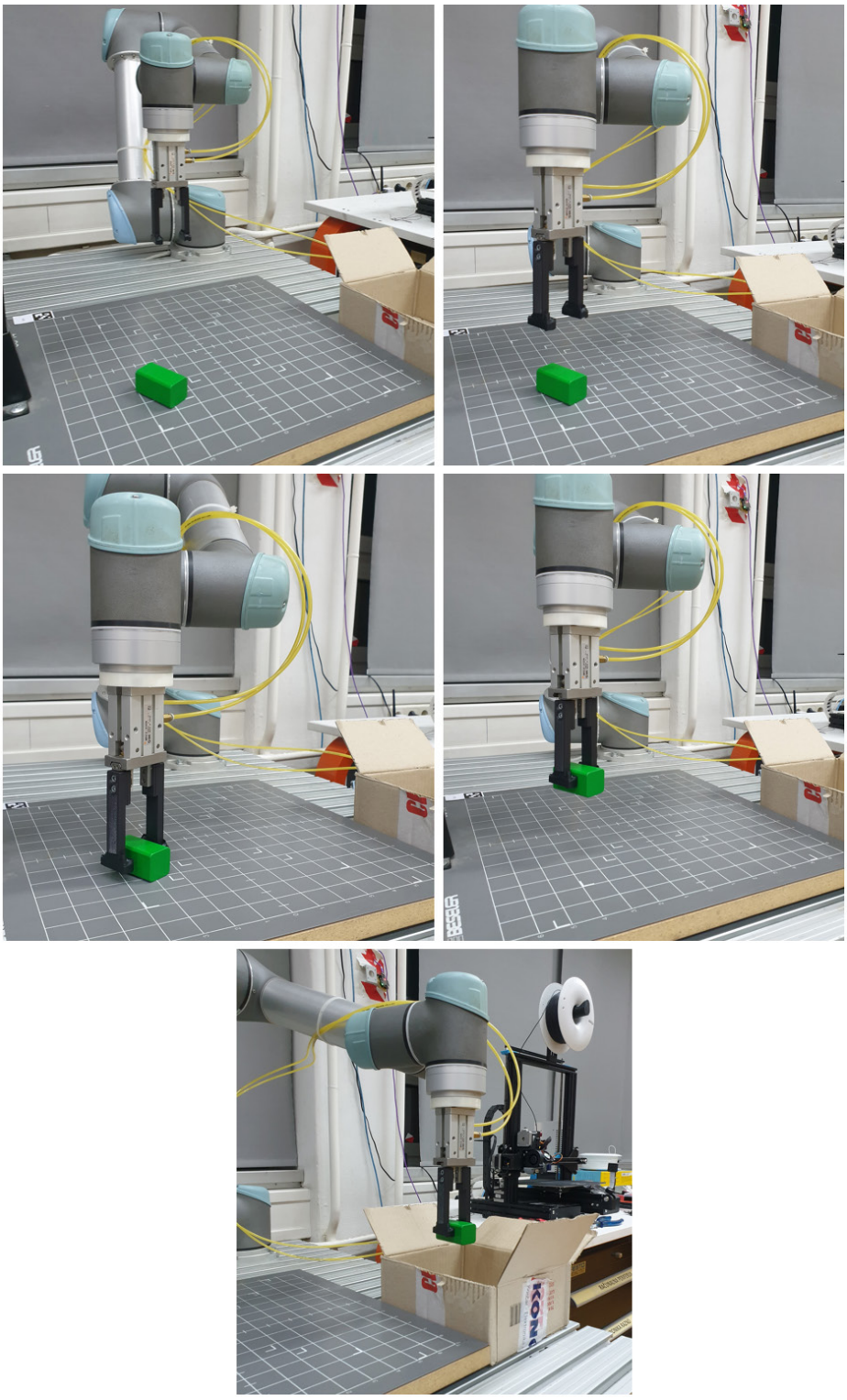
Slika 7.5. Koordinate vrha hvataljke iznad odabranog ishodišta

Nakon što su poznate koordinate ishodišta u koordinatnom sustavu robota moguće je poslati robota u tražene točke na kojima program prepoznaje položaj predmeta relativnim pomakom u smjeru osi x i y u odnosu na definirano ishodište. To je izvršeno na način da je udaljenost predmeta od ishodišta po osi x spremljena u globalnu varijablu pozicija_x, udaljenost predmeta od ishodišta po osi y spremljena je u globalnu varijablu pozicija_y, dok je iznos zakreta spremljen u globalnu varijablu naziva zakret. Pozivom programa prikazanog na slici (Slika 7.6) omogućeno je robotsko izuzimanje traženog predmeta.

```
rob = urx.Robot ("192.168.0.25")
rob.set_tcp((0, 0, 0.167, 0, 0, 0))
rob.set_payload(0.5, (0, 0, 0))
rob.set_digital_out(0,0)
v = 0.05
a = 0.3
pozicija = [-0.2855527, 0.3870877, 0.1011059, -2.2251891, 2.2101920, -0.0179196]
pozicija[0]-=pozicija_x
pozicija[1]-=pozicija_y
rob.movel(pozicija, acc=a, vel=0.3, wait=True)
kut = rob.getj()
kut[5]+=zakret
rob.movej(kut, acc=a, vel=0.4, wait=True)
hvatanje = rob.getl()
hvatanje[2] = 0.0324173
rob.movel(hvatanje, acc=a, vel=v, wait=True)
rob.set_digital_out(0,1)
time.sleep(1)
hvatanje[2]=0.1
rob.movel(hvatanje, acc=a, vel=v, wait=True)
kutija=[3.44663, -1.77134, -1.73639, -1.21471, 1.57845, 3.47790]
rob.movej(kutija, acc=a, vel=0.5, wait=True)
rob.set_digital_out(0,0)
rob.close()
```

Slika 7.6. Program za izuzimanje predmeta

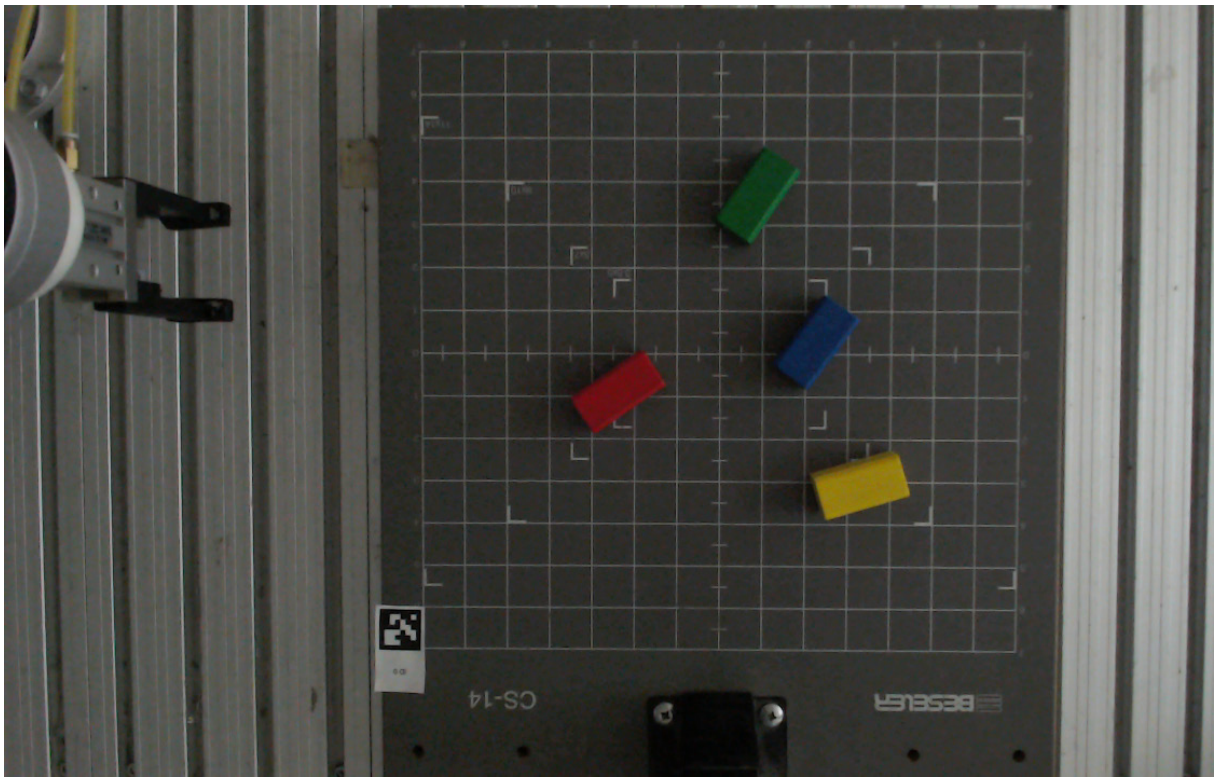
Slika 7.7 prikazuje korake robotskog izuzimanja. U prvom koraku hvataljka se pozicionira iznad predmeta preko koordinata koje je program izračunao, a vidljive su na slici (Slika 7.2). Nakon toga slijedi rotacija hvataljke u ovisnosti o zakretu predmeta. Potom se hvataljka spušta po z osi, nakon čega se vrši zatvaranje hvataljke.



Slika 7.7. Robotsko izuzimanje

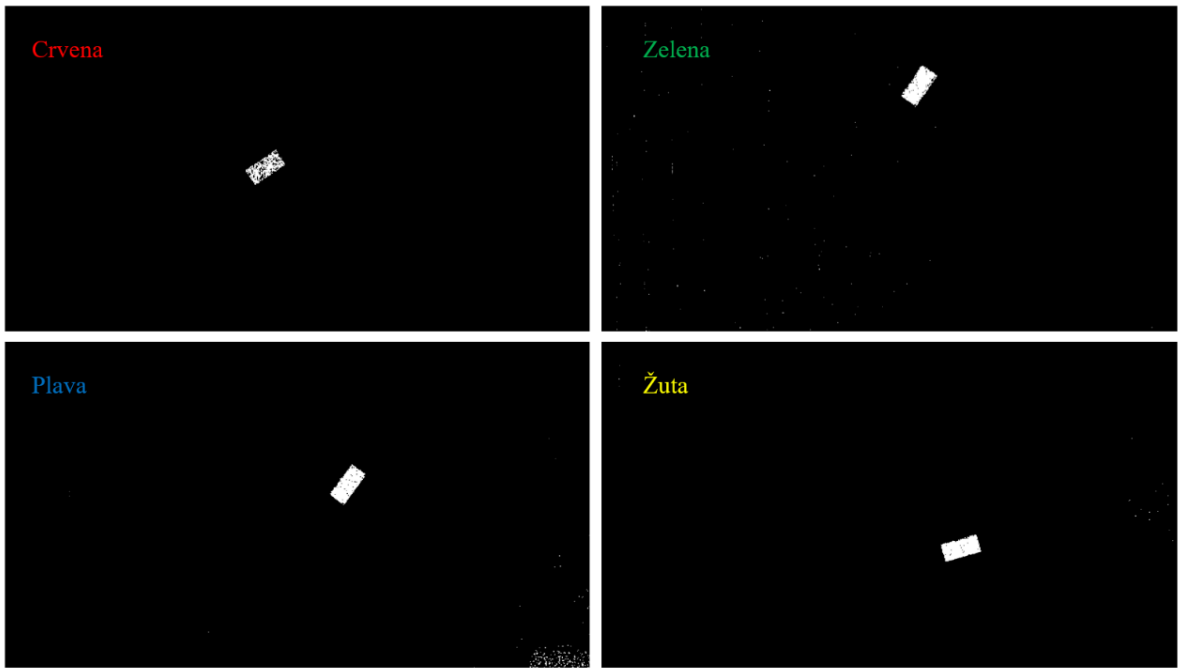
Na robotskoj kontrolnoj jedinici hvataljka je spojena na digitalni izlaz 0. Pozivom funkcije rob.set_digital_out mijenjaju se vrijednosti pojedinih izlaza. U konkretnom slučaju vrijednost 0 označava otvorenu hvataljku, dok je kod vrijednosti 1 hvataljka zatvorena. Nakon što je hvataljka zatvorena, robot se podiže po z osi i dovodi u poziciju iznad kutije gdje se vrši odlaganje predmeta.

Nakon što je utvrđeno da programska aplikacija pruža odgovarajuće rezultate pri dobrim svjetlosnim uvjetima potrebno je provjeriti kako program reagira na uvjete slabijeg osvjetljenja. Slika 7.8 prikazuje izlaznu snimku programa u tamnim uvjetima u prostoriji.



Slika 7.8. Uvjeti slabijeg osvjetljenja

S obzirom da prethodno postavljena maska u ovom slučaju ne može očitati predmete potrebno je promijeniti parametre nijanse, zasićenosti i svjetlosti za novonastale uvjete. Nakon postavljanja novih vrijednosti rezultati odstupaju u odnosu na uvjete dobrog osvjetljenja. Na slici (Slika 7.9) može se vidjeti kako je maska pojedine boje prikazana u tamnim uvjetima. U slučaju zelene, žute i plave boje maska je relativno dobro formirana, ali se na prikazu pojavljuju područja s malim vizualnim smetnjama. Tih smetnji nema na prikazu crvene maske, ali je promatrani objekt u tom slučaju najslabije prepoznat. Iz tog razloga je aplikaciju najbolje koristiti u uvjetima dobrog osvjetljenja pri nepromjenjivom intenzitetu izvora svjetlosti.



Slika 7.9. Maska u tamnim uvjetima

8. ZAKLJUČAK

U okviru ovog diplomskog rada opisana je izrada programske aplikacije za robotsko izuzimanje objekata temeljem boje i oblika. Kako bi se to ostvarilo korišten je računalni vid izveden preko web kamere koja je poslužila kao vizijski senzor koji dohvaća sliku iz okoline u stvarnom vremenu. Na početku rada dana je teorijska podloga koja sadrži osnovne informacije o područje umjetne inteligencije koje obuhvaća računalni vid te su opisane metode pomoću kojih je izvršeno prepoznavanje objekata. Također je opisan programski jezik Python kojim je napisana programska aplikacija te knjižnice koje su korištene u izradi aplikacije. Korištena metoda za segmentaciju slike je metoda maskiranja koja se pokazala pogodnom za korištenje u stvarnom vremenu. Korišteni algoritmi za prepoznavanje objekata dio su knjižnice računalnog vida OpenCV. Nakon što je objekt traženih atributa prepoznat bilo je potrebno izvršiti lokalizaciju i odrediti orijentaciju objekta kako bi se moglo provesti robotsko izuzimanje. Napravljen je sustav koji se sastoji od proizvoljno postavljenog ishodišta izvedenog putem ArUco markera pomoću kojeg se izvršava pozicioniranje robotske hvataljke u ovisnosti o koordinatama pronađenog objekta unutar radnog prostora. Robot korišten za izuzimanje objekata je Universal Robots UR5, a komunikacija između robota i računala ostvaruje se TCP protokolom. Robotska hvataljka potrebna za izuzimanje predmeta oblikovana je putem CAD programskog alata Inventor i izrađena korištenjem aditivne tehnologije 3D ispisivanja. Analizom rezultata razvijenog sustava u različitim uvjetima osvjetljenja zaključeno je da se u slučaju promjene osvjetljenja može izvršiti promjena parametara maske. Međutim, radi ostvarenja optimalnih rezultata aplikaciju je zamišljeno koristiti pri dobrim svjetlosnim uvjetima uz konstantni intenzitet izvora svjetlosti. Aplikaciju je moguće upotrijebiti za rad s nizom proizvoda različitih oblika i boja u industriji. Daljnji razvoj aplikacije može ići u smjeru korištenja neuronskih mreža radi povećanja opsega predmeta koje je moguće prepoznati. Također je moguće zamijeniti oblikovanu hvataljku vakuumskom radi veće fleksibilnosti kod rada s predmetima složenije geometrije.

LITERATURA

[1] Pulli K, Baksheev A, Kornyakov K, Eruhimov V. Realtime Computer Vision with OpenCV. ACM Queue. 2012;10(4):40–56. https://doi.org/10.1145/2181796.2206309

[2] https://viso.ai/deep-learning/object-detection/, Pristupljeno: 15.10.2022.

[3] https://www.python.org/about/, Pristupljeno: 16.10.2022.

[4] https://opencv.org/about/, Pristupljeno: 16.10.2022.

[5] https://www.edureka.co/blog/python-libraries/, Pristupljeno: 16.10.2022.

[6] https://github.com/SintefManufacturing/python-urx, Pristupljeno: 16.10.2022.

[7] https://en.wikipedia.org/wiki/Color_model, Pristupljeno: 22.10.2022.

[8] https://en.wikipedia.org/wiki/RGB_color_model, Pristupljeno: 22.10.2022.

[9] https://docs.aspose.com/html/net/tutorial/html-colors/, Pristupljeno: 22.10.2022.

[10] https://en.wikipedia.org/wiki/CMYK_color_model, Pristupljeno: 22.10.2022.

[11] https://www.123rf.com/photo_65990122_color-colors-wheel-names-degrees-rgb-hsb-hsv-hue.html, Pristupljeno: 22.10.2022.

[12] https://en.wikipedia.org/wiki/HSL_and_HSV, Pristupljeno: 25.10.2022.

[13] https://codeloop.org/opencv-python-color-detection-example/, Pristupljeno: 25.10.2022.

[14] https://en.wikipedia.org/wiki/Ramer%E2%80%93Douglas%E2%80%93Peucker_algorithm, Pristupljeno: 26.10.2022.

[15] https://towardsdatascience.com/simplify-polylines-with-the-douglas-peucker-algorithm-ac8ed487a4a1, Pristupljeno: 26.10.2022.

[16] https://learnopencv.com/find-center-of-blob-centroid-using-opencv-cpp-python/, Pristupljeno: 1.11.2022.

[17] https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html, Pristupljeno: 1.11.2022.

[18] https://clickitupanotch.com/lens-distortion/, Pristupljeno: 2.11.2022.

[19] https://learnopencv.com/camera-calibration-using-opencv/, Pristupljeno: 5.11.2022.

[20] https://aliyasineser.medium.com/opencv-camera-calibration-e9a48bdd1844, Pristupljeno: 5.11.2022.

[21] https://en.wikipedia.org/wiki/Universal_Robots, Pristupljeno: 12.11.2022.

[22] https://robots.ieee.org/robots/universal/, Pristupljeno: 12.11.2022.

[23] Kufieta, K.: Force Estimation in Robotic Manipulators: Modeling, Simulation and Experiments, NTNU Norwegian University of Science and Technology, 2014.

[24] https://en.wikipedia.org/wiki/Transmission_Control_Protocol, Pristupljeno: 14.11.2022.

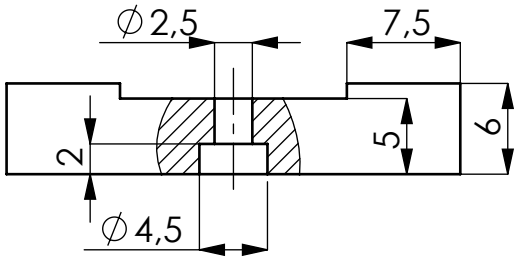
[25] https://www.sciencedirect.com/topics/computer-science/three-way-handshake, Pristupljeno: 14.11.2022.

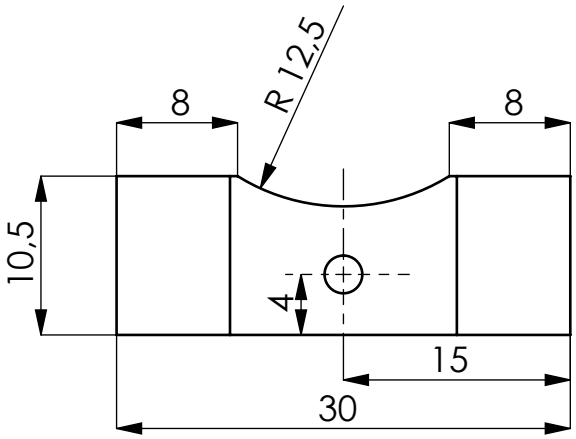
[26] https://en.wikipedia.org/wiki/3D_printing, Pristupljeno: 14.11.2022.

[27] https://en.wikipedia.org/wiki/STL_(file_format), Pristupljeno: 14.11.2022.

PRILOZI

I. Tehnička dokumentacija

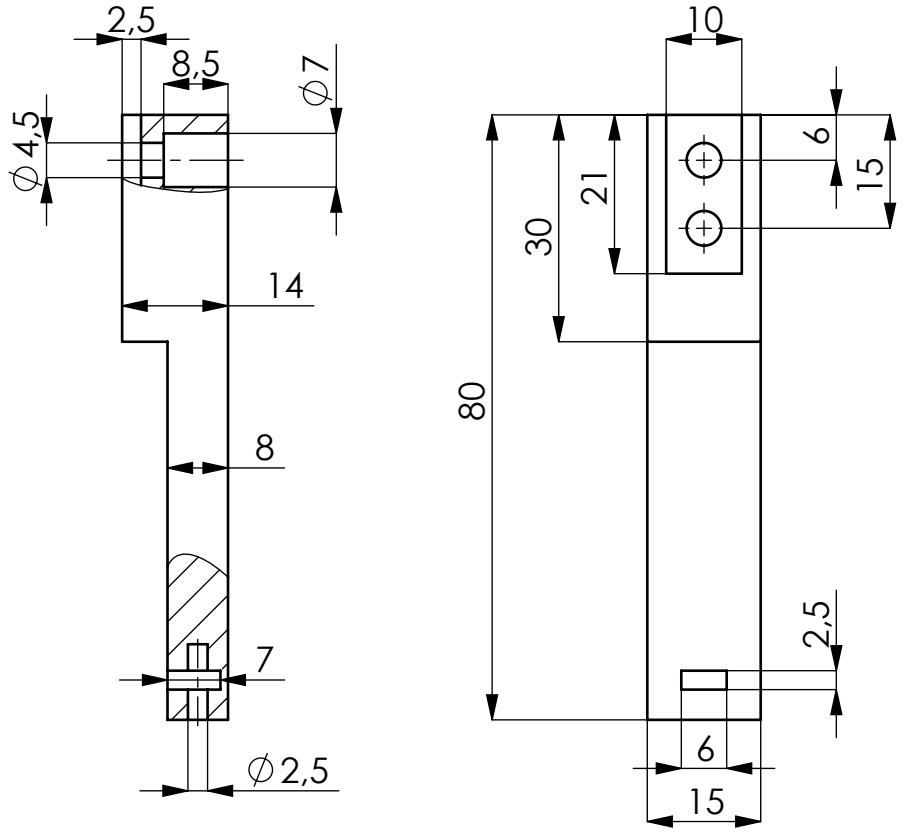




	Datum	Ime i prezime	Potpis
Projektirao	15.11.2022.	Luka Sinjeri	
Razradio	20.11.2022.	Luka Sinjeri	
Crtao			
Pregledao			

FSB Zagreb

Objekt:	Objekt broj:
	R. N. broj:
Napomena:	Kopija
Materijal: PLA	Masa:
Mjerilo originala 2:1	Naziv: Nastavak hvataljke
Pozicija:	Format: A4
Listova: 1	List: 1
Crtež broj: LS-DR-01	



	Datum	Ime i prezime	Potpis
Projektirao	16.11.2022.	Luka Sinjeri	
Razradio	20.11.2022.	Luka Sinjeri	
Crtao			
Pregledao			

FSB Zagreb

Objekt:	Objekt broj:
	R. N. broj:
Napomena:	Kopija
Materijal: PLA	Masa:
Mjerilo originala 2:1	Naziv: Prsti hvataljke
Pozicija:	Format: A4
Listova: 1	List: 1
Crtež broj: LS-DR-02	

Design by CADLab