Rad robota na temelju prepoznatih objekata snimljenih kamerom

Hmelik, Marko

Undergraduate thesis / Završni rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University North / Sveučilište Sjever**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:122:335105

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-02-24**

Repository / Repozitorij:

University North Digital Repository

Sveučilište
Sjever

Završni rad br. 037/MEH/2024

Rad robota na temelju prepoznatih objekata snimljenih kamerom

Marko Hmelik, 0336048965

Varaždin, rujan 2024. godine

Sveučilište
Sjever

Odjel za mehatroniku

Završni rad br. 037/MEH/2024

Rad robota na temelju prepoznatih objekata snimljenih kamerom

Student

Marko Hmelik, 0336048965

Mentor

Zoran Busija, dipl.ing.stroj.

Varaždin, rujan 2024. godine

Sveučilište Sjever
Sveučilišni centar Varaždin
104. brigade 3, HR-42000 Varaždin

Prijava završnog rada

Definiranje teme završnog rada i povjerenstva

ODJEL Odjel za mehatroniku

STUDIJ preddiplomski stručni studij Mehatronika

PRISTUPNIK Marko Hmelik | JMBAG 0336048965

DATUM 09.09.2024. | KOLEGIJ Robotika

NASLOV RADA Rad robota na temelju prepoznatih objekata snimljenih kamerom

NASLOV RADA NA ENGL. JEZIKU The operation of the robot based on the recognized objects recorded by the camera

MENTOR Zoran Busija, dipl. ing. stroj. | ZVANJE predavač

ČLANOVI POVJERENSTVA

1. Siniša Švoger, mag.ing.mech, predavač
2. prof. dr. sc. Ante Čikić
3. Zoran Busija, dipl.ing.stroj, predavač
4. Josip Srpak, dipl.ing.el, viši predavač
5.

Zadatak završnog rada

BROJ 037/MEH/2024

OPIS

U završnom radu potrebno je:
- pojasniti načine detekcije objekata na slici.
- razraditi sustav za prepoznavanje objekata pomoću kamere koji će komunicirati s robotom.
- opisati pojedine softverske elemente korištene u radu
- prikazati izradu programa u Python-u.
- napraviti simulaciju rada u softveru RobotStudio.
- testirati rad između Python programa i softvera RobotStudio.

ZADATAK URUČEN 12.09.2024. | POTPIS MENTORA Busija Zoran

REPUBLIKA HRVATSKA SVEUČILIŠTE SJEVER

Predgovor

Ideju završnog rada dobio sam iz znatiželje o tehnologijama umjetne inteligencije. Prije izrade ovog rada, način učenja računala i pojam neuronske mreže u smislu računalnog učenja, bile su „crna kutija“ koja se činila jako zanimljiva i fascinantna. Ovaj rad je bila dobra prilika da napravim prvi korak u razumijevanju ovih tehnologija, a povezivanje rada robota je rad napravilo još zanimljivijim. Sam proces izrade funkcionalnog programa smatram jako zadovoljavajućim, kao i proces automatiziranja nekog procesa, a u ovom radu imao sam priliku povezati više programa u automatiziranu cjelinu.

Zahvaljujem se mentoru Zoranu Busiji, dipl. ing. stroj. na prenesenom znanju tijekom studiranja te na pomoći, utrošenom vremenu i strpljenju pri izradi završnoga rada.

Hvala obitelji, djevojci i prijateljima na podršci i motivaciji tijekom studiranja, te ostalim profesorima Sveučilišta Sjever na znanju prenesenom tijekom studiranja.

Marko Hmelik

Sažetak

U ovome radu je u Python programu treniran YOLOv8 model neuronske mreže da prepozna čovjeka, model kocke i valjka na kameri. Zatim je u drugom Python programu napravljen server za *socket* komunikaciju TCP protokola, pomoću koje se informacije o detektiranim objektima u kadru kamere šalju u RAPID zadatak. U Python-u je također napravljen program sučelja koji objedinjuje program za detekciju i program za *socket* komunikaciju u jednu cjelinu.

Program RobotStudio poslužio je za izradu okruženja robota. Također je napravljena simulacija pomoću značajki koje će biti detaljno opisane u ovom radu. Modeli korišteni za simulaciju napravljeni su u SolidWorks programu. U RAPID programskom jeziku programa RobotStudio programiran je klijent za *socket* komunikaciju koji stalno komunicira sa Python programom. U RAPID-u se također upravlja radom robota i signalima kontrolera.

Ključne riječi: YOLOv8, RobotStudio, RAPID, *socket*, SolidWorks

Abstract

In this paper, a YOLOv8 neural network model was trained in Python to recognize a human, a cube and a cylinder model on a camera. Then, in another Python program, a server for TCP protocol socket communication was created, by means of which information about detected objects in the camera frame is obtained and sent to the RAPID task. An interface program was also created in Python that combines the detection program and the socket communication program into a single entity. The RobotStudio program was used to create the robot environment. A simulation was also made using the features that will be described in detail in this paper. The models used for the simulation were made in the SolidWorks program. In the RAPID programming language, the RobotStudio program is programmed as a socket communication client that constantly communicates with the Python program. Robot operation and controller signals are also managed in RAPID.

Keywords: YOLOv8, RobotStudio, RAPID, socket, SolidWorks

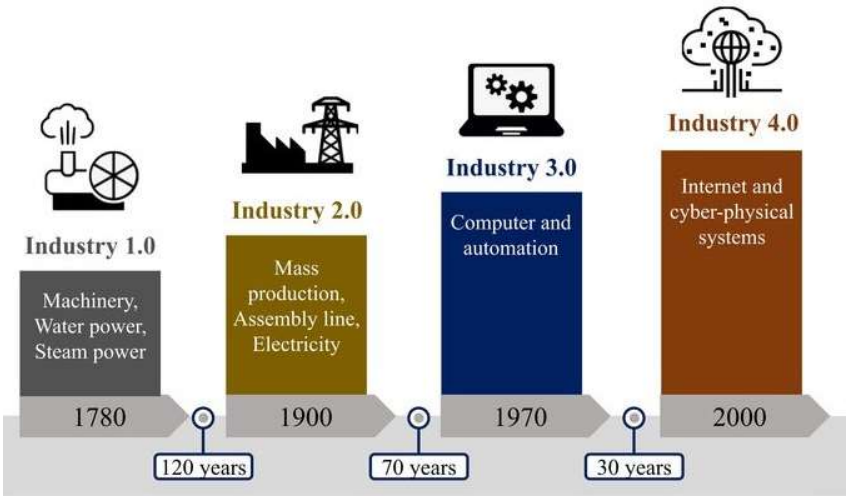
Popis korištenih kratica

CNN	Convolutional neural network konvolucijske neuronske mreže
RNN	Recurrent neural networks Rekurentne neuronske mreže
YOLO	You Only Look Once „Pogledaš samo jednom„ - model neuronske mreže
TP	True positive pozitivne pretpostavke
FP	False positive netočno pozitivne pretpostavke
FN	False negative netočno negativne pretpostavke
IoU	intersection over union presjek preko unije
TCP	Transfer Control Protocol Protokol kontrole prijenosa
UDP	User Datagram Protocol Protokol korisničkog *datagrama*
IP	Internet Protocol Internet Protokol
FPS	Frames per second Slike u sekundi
ASCII	American Standard Code for Information Interchange Američki standardni kod za razmjenu informacija
TCP	Tool Central Point Središnja točka alata

Sadržaj

1. Uvod

Gledajući razvoj tehnologije, može se zaključiti da je vremenski razmak između velikih tehnoloških napredaka sve manji i manji (slika 1.1).



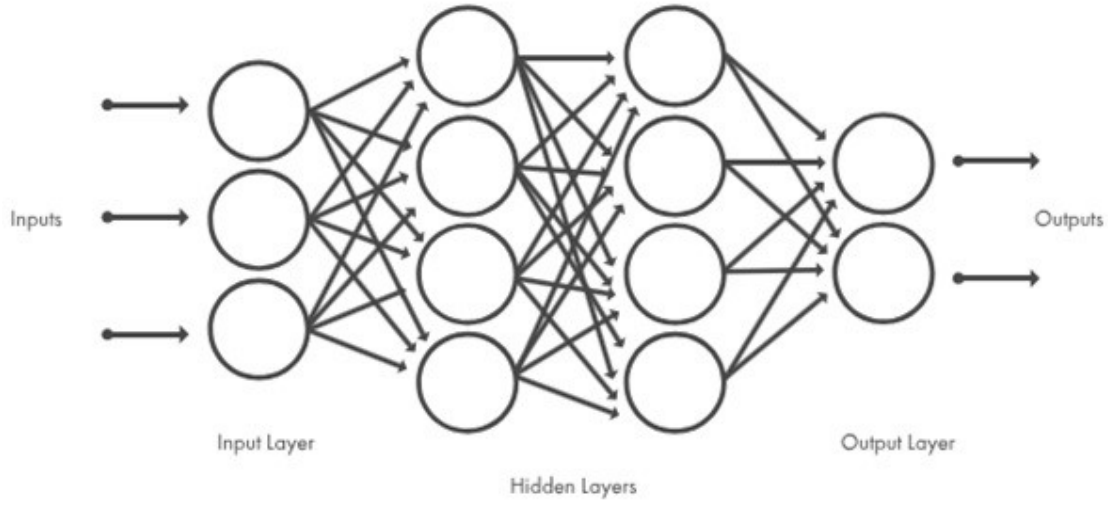
Slika 1.1 – Vremensko razdoblje industrijskih revolucija [1]

Točan datum početka industrije 4.0 nije utvrđen, ali činjenica je da se nalazimo u razdoblju četvrte industrijske revolucije koju karakteriziraju AI (eng. artificial intelligence), strojno i dubinsko učenje, strojni vid, te IoT (eng. Internet of things) i ostale tehnologije. AI tehnologija postaje sve učestalija, a do kraja desetljeća će vjerojatno postati neizbježan dio svakodnevnog života. Prepoznavanje objekata već se koristi u svakodnevnom životu u područjima poput prepoznavanja lica na pametnim telefonima, ali i sigurnosnim kamerama, kontroli sigurnosti, u jednostavnim zadacima u industriji poput klasifikacije i brojanja objekata te u analizi u medicini.

Cilj ovoga rada je napraviti program za prepoznavanje objekta na slici, a zatim tu informaciju prenijeti robotu. Robot na temelju te informacije izvršava pokupi i stavi (eng. pick and place) zadatak. Dok robot izvršava taj zadatak, u stvarnom vremenu se na kameri prati prisutnost čovjeka. Čim se čovjek pojavi u kadru kamere, robot mora zaustaviti svo kretanje sve dok se čovjek ne makne iz kadra kamere. Ovo predstavlja kontrolu sigurnosti u robotskom okruženju u kojem čovjek ne smije biti u prisustvu robota dok se izvršava rad.

2. Dubinsko učenje

Duboko učenje je specijalizirani oblik strojnog učenja, a oba su dio područja umjetne inteligencije (AI). Arhitekture neuronskih mreža su temelji modela dubinskog učenja. Ove mreže, koje uzimaju inspiraciju iz ljudskog mozga, sastoje se od slojeva međusobno povezanih neurona koji povezuju ulaze sa željenim izlazima. Između ulaznog i izlaznog sloja nalaze se skriveni slojevi, a količina tih slojeva određuje „dubinu“ mreže. Izraz „dubinsko“ obično se koristi za opisivanje neuronskih mreža s brojnim skrivenim slojevima. Oni mogu varirati od stotina do tisuća u modelima dubinskog učenja. Na slici 2.1. vidi se arhitektura tipične neuronske mreže. Modeli dubokog učenja obučavaju se korištenjem velikih skupova označenih podataka [2].



Slika 2.1.- Arhitektura tipične neuronske mreže [2]

Tri glavna tipa modela neuronskih mreža dubinskog učenja se dijele na:

1. Konvolucijske neuronske mreže - CNN (eng. convolutional neural networks),
2. Rekurentne neuronske mreže – RNN (eng. recurrent neural networks),
3. Modeli transformatora (eng. transformer models).

CNN arhitektura dubinskog učenja, koja je temelj za model korišten u praktičnom dijelu ovog rada, prikladna je za obradu 2D podataka, kao što su slike. Ova arhitektura radi na način da izdvaja značajke izravno iz slika. Bitne značajke se nauče dok mreža uči na zbirci slika. Ova automatizirana ekstrakcija značajki čini modele dubinskog učenja vrlo preciznim za zadatke klasifikacije slika, no ova arhitektura se koristi i za klasifikaciju drugih vrsta podataka poput teksta [2].

2.1. Detekcija objekata

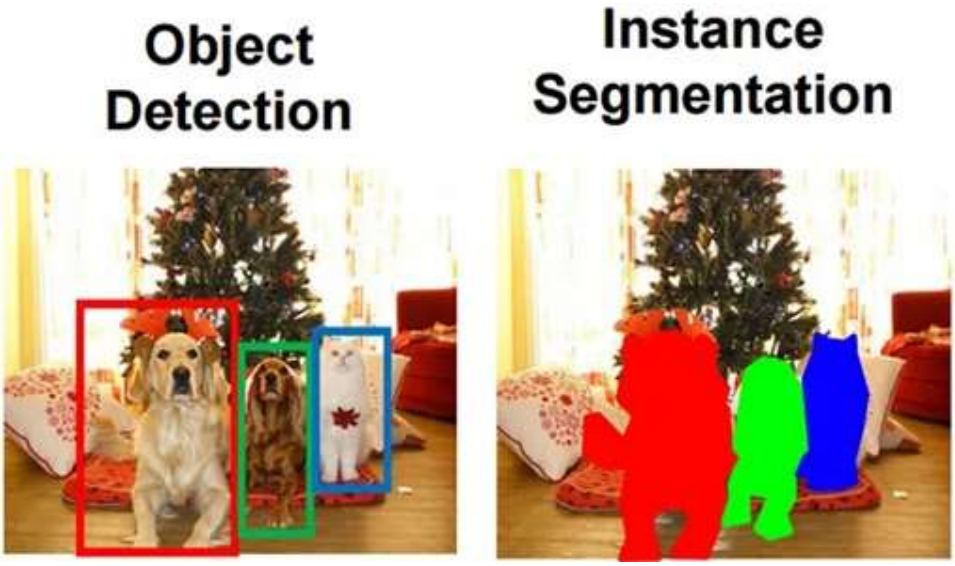
Detekcija objekata je zadatak koji se temelji na računalnom vidu. Detekcija objekata je danas sve rasprostranjenija te se može naći u područjima poput robotike i nadziranja. Detekcija na temelju slike predviđa o čemu se na slici radi. Na temelju broja prolaza ulazne slike kroz mrežu, detekcija se može podijeliti na dvije glavne skupine:

1. Detekcija s jednom fazom,
2. Detekcija s dvije faze.

Kod detekcije s jednom fazom, slika jednom prolazi kroz mrežu, te se na temelju toga stvara pretpostavka o prisustvu i lokaciji objekta na slici. Ovaj način detektiranja nije efikasan kao ostale metode. Kod detekcije s dvije faze, slika dva puta prolazi kroz mrežu. U prvom prolazu se stvaraju prijedlozi o lokaciji i prisustvu objekta, a u drugom prolazu se ti prijedlozi potvrđuju. Ovaj način detektiranja efikasniji je od detekcije sa jednom fazom [3].

2.2. YOLOv8

Što je YOLO? Ime YOLO dolazi od skraćenice „pogledaš samo jednom“ (eng. you only look once). Kao što ljudskom oku treba jedan pogled da prepozna neki objekt, tako je i ovom modelu neuronske mreže cilj na „prvi pogled“ prepoznati objekt sa što većom sigurnošću te mu dodijeliti okvir i pripadajuće ime. YOLOv8 je model neuronske mreže za klasifikaciju, detekciju i segmentaciju u stvarnom vremenu. Tvrtka Ultralytics, među ostalim alatima za strojno učenje, održava i razvija YOLO seriju modela neuronske mreže. YOLO model mreže temelji se na CNN tipu dubinskog učenja modela neuronskih mreža, a detekcija objekata spada pod detekciju s jednom fazom. Dakle, detekcija se odnosi na definiranje prisustva objekta na slici, te na određivanje njegove lokacije pomoću okvira. Segmentacija također određuje prisustvo objekta na slici, ali za razliku od detekcije, puno je preciznija u određivanju lokacije objekta. Umjesto korištenja klasičnog okvira oblika pravokutnika, objekti se kod segmentacije odjeljuju točno do piksela. Razlika između detekcije i segmentacije prikazana je na slici 2.2 [4].



Slika 2.2.- Prikaz detekcije i segmentacije na istom primjeru [5]

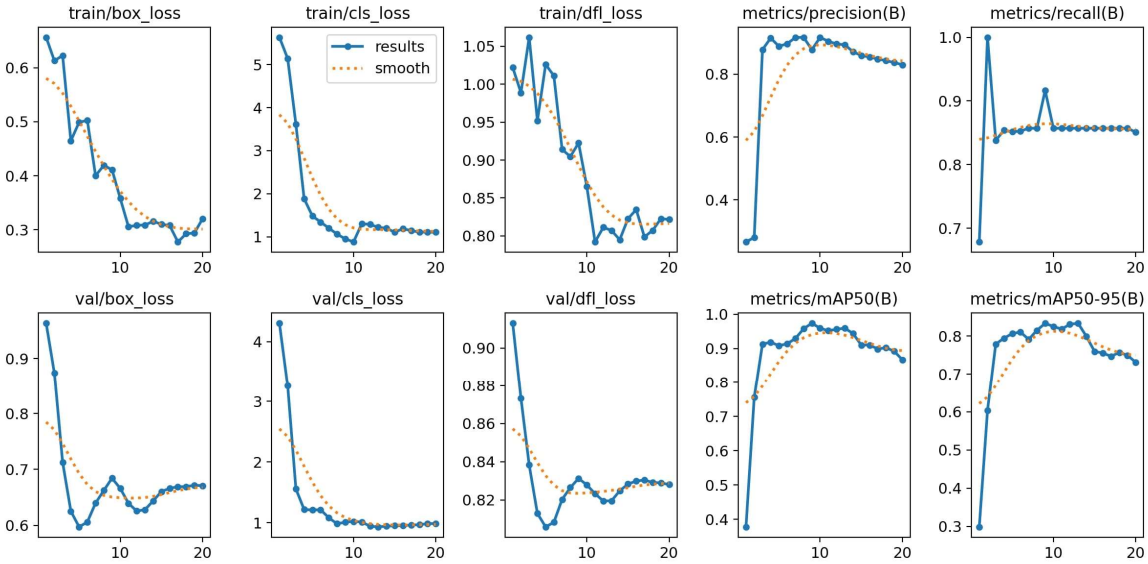
Kod klasifikacije, za svaku sliku se stvara jedna oznaka. Za razliku od detekcije gdje se na istoj slici može detektirati više objekata, klasifikacija može sliku svrstati samo pod jednu oznaku. U ovom radu koristiti će se isključivo detekcija.

2.2.1. Način učenja YOLOv8 modela

Za učenje modela, YOLOv8 koristi velik broj slika sa oznakama. Slike i njima pripadajuće oznake moraju biti istoga imena. Slike su u *.JPG formatu, a oznake u *.txt formatu. Svaki objekt, odnosno klasa, ima svoj identifikacijski broj. Slike i oznake se svrstavaju na podatke za trening i validaciju. Model direktno uči iz podataka za trening. Nakon svake iteracije treninga, model napravi pretpostavku koja uključuje lokaciju i prisustvo okvira, identifikaciju klase objekta na slici te povjerenje (eng. confidence) u dane pretpostavke. Podaci o validaciji koriste se za ocjenjivanje učinka modela mreže tijekom obuke te pomažu u praćenju sposobnosti modela na novim, neviđenim, podacima.

2.2.2. Rezultati učenja YOLOv8 modela

Nakon svakog učenja, rezultati pojedinih parametara učenja modela neuronske mreže prikazuju se grafovima (slika 2.3). Ovi grafovi se nalaze na slikama koje su generirane na kraju treninga i spremljene u mapi sa ostalim rezultatima treninga.



Slika 2.3.- Primjer parametara učenja YOLOv8 modela neuronske mreže

Svaki graf pokazuje jedan parametar u vremenu, odnosno, mijenjanje parametara tijekom iteracija učenja. Parametri sa prefiksom „train“ odnose se na podatke korištene za trening, koji su stavljeni u mapu „train“. Parametri sa prefiksom „val“ odnose se na podatke korištene za validaciju, koji su stavljeni u mapu nazvanu „val“. Parametar box_*loss* označava netočnost pretpostavljene lokacije i veličine okvira objekta. Poželjno je vidjeti da graf ovog parametra pada, što indicira povećanje točnosti kod predviđanja okvira objekata. Parametar cls_loss označava netočnost kod pretpostavljanja klase objekta. Kod ovoga dijagrama je također poželjno vidjeti pad koji označava poboljšanje kod klasifikacije objekta. Parametar dfl_loss (eng. distribution focal loss) služi za preciznije predviđanje pozicija okvira. Kod grafa ovog parametra je također poželjan pad, koji označuje da je model mreže bolji u preciznom određivanju okvira za objekte. Ako ovi dijagrami prestanu padati, ali i rasti, to indicira da je model mreže postigao vrhunac svog kapaciteta za učenje. Za bolje rezultate bi se trebalo dodati više podataka, među kojima postoji varijacija. Ako ovi grafovi počnu rastu, to indicira pojavu zvanu „Overfitting“, pri kojem model mreže postane izrazito dobar u predviđanju kod podataka za trening i/ili validaciju. Ali to je postigao pamćenjem jako specifičnih uzoraka i šumova, pa zbog toga postaje sve lošiji u predviđanju novih i neviđenih primjeraka. To se može riješiti dodavanjem novih podataka među kojima postoji znatna varijacija poput kuta slike, osvjetljenja, boja na slici, udaljenosti i slično. Parametar preciznost (eng. precision) predstavlja pozitivne pretpostavke tijekom treninga, a računa se formulom:

$$Preciznost = \frac{TP}{TP + FP}$$

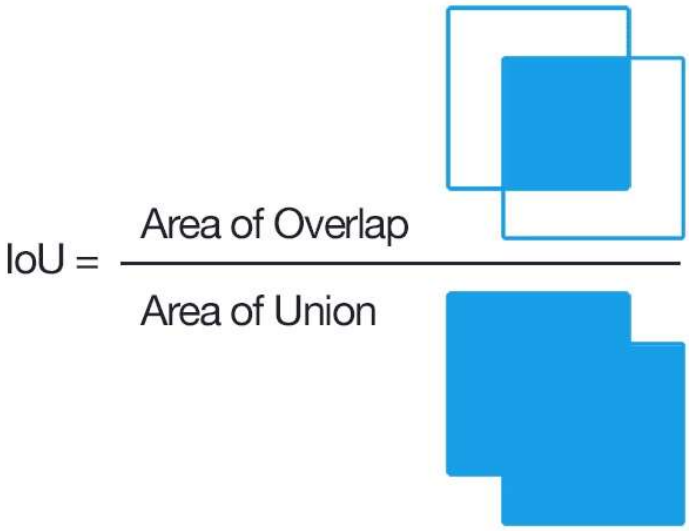
gdje TP označava točno pozitivne pretpostavke (eng. true positives), a FP netočno pozitivne pretpostavke (eng. false positives). Dakle, dobro je ako graf ovoga parametra raste. Parametar podsjećanja (eng. recall) označava sposobnost modela mreže da detektira samo objekte koje treba detektirati. Računa se formulom:

$$Podsjećanje = \frac{TP}{\mathrm{TP} + \mathrm{FN}}$$

gdje TP označava točno pozitivne pretpostavke (eng. true positives), a FN netočno negativne pretpostavke (eng. false negatives). Ovaj graf bi također trebao rasti. Parametar mAP (eng. mean Average Precision) uzima parametar preciznosti i parametar podsjećanja i spaja ih u cjelinu izračunavanjem prosječne preciznosti u svim klasama te uzimajući srednju vrijednost tih brojeva. Parametar mAP50 je vrijednost mAP, ali sa uvjetom da je vrijednost IoU (eng. intersection over union) 0.5. Parametar IoU (slika 2.4) se računa formulom:

$$IoU = \frac{AoO}{\mathrm{AoU}}$$

Gdje AoO (eng. Area of Overlap) predstavlja područje preklapanja, a AoU područje unije. Ova su područja prikazana na slici 2.4.



Slika 2.4. – Prikaz područja preklapanja i unije [6]

Dakle, mAP50 je mjera točnosti koja u obzir uzima lagane detekcije jer detekcija treba imati samo 50% IoU. Parametar mAP50-95 je isti kao mAP50, ali u obzir uzima vrijednosti IoU od 0.5 do 0.95. Ovime pokriva veći domet težine detektiranja jer je teže dobiti detekcije sa većom IoU

vrijednosti. Zbog toga je normalno da mAP50-95 bude niži od parametra mAP50. Rastući graf je poželjan kod oba od navedenih parametara [7] [8].

3. Softverske komponente sustava

3.1. Python

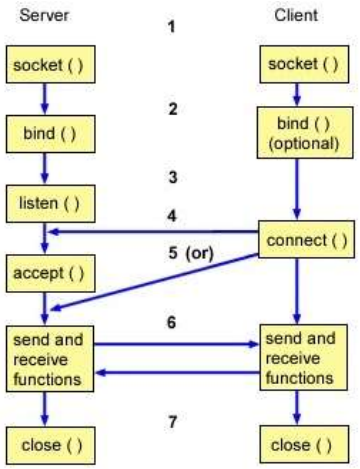
Python je programski jezik visoke razine te je u ovome radu korišten za treniranje modela neuronske mreže, komunikaciju sa RobotStudio-m te izradu sučelja. Python obilježavaju jednostavnost, fleksibilnost i naglašena čitljivost. Sintaksa je jednostavna za razumjeti, a tipovi varijabla se ne moraju eksplicitno deklarirati zbog značajke „Dynamic Typing“. Python je fleksibilan programski jezik i koristi se u razne svrhe poput razvoja web-a, strojnog učenja, automatizacije… Python sadrži opsežnu biblioteku, no u svrhu ovoga zadatka, bilo je potrebno uvesti nove biblioteke [9][10].

3.2. Pytorch

PyTorch je okvir (eng. framework) za dubinsko učenje, kojega je razvio laboratorij Facebook AI Istraživanje (eng. Facebook AI Research - FAIR). Koristi se za zadatke strojnog učenja poput računalnog vida i obrade prirodnog jezika. PyTorch pruža Python paket sa značajkama za računalne operacije tenzora, te nudi podršku grafičke kartice. Ovo je važno jer treniranje modela neuronskih mreža sadrži veliki broj operacija matricama i matematičkih računanja te se korištenjem grafičke kartice, umjesto samo procesora, značajno ubrzava proces treniranja [11][12].

3.3. Socket komunikacija

Socket je točka komunikacijske veze koju se može imenovati i adresirati u mreži. *Socket*-i omogućuju razmjenu informacija između procesa na istom računalu ili preko mreže. *Socket*-i se obično koriste za interakciju klijenta i poslužitelja. Tipična konfiguracija sustava postavlja poslužitelj na jedno računalo, a klijente na druga računala. Klijenti se spajaju na poslužitelj, razmjenjuju informacije i zatim prekidaju vezu. *Socket* komunikacija ima slijedeći tok događaja (slika 3.1). U modelu klijent-poslužitelj, *socket* na poslužiteljskom procesu čeka zahtjeve od klijenta. Da bi to učinio, poslužitelj prvo uspostavlja adresu koju klijenti mogu koristiti za pronalaženje poslužitelja. Kada je adresa uspostavljena, poslužitelj čeka da klijenti zatraže uslugu. Razmjena podataka između klijenta i poslužitelja odvija se kada se klijent poveže s poslužiteljem putem *socket*-a. Poslužitelj izvršava zahtjev klijenta i šalje odgovor nazad klijentu [13] [14].



Slika 3.1.- Tijek događaja socket komunikacije [13]

Dvije glavne vrste *socket*-a su „Stream Sockets“ koji koriste TCP (eng. Transfer Control Protocol) protokol i „Datagram Sockets“ koji koriste UDP (eng. User Datagram Protocol) protokol. „Stream Sockets“ vrsta *socket*-a pruža serijski, konstantan i pouzdan dvosmjerni protok podataka. Nakon uspostave veze, podaci se čitaju i pišu u nizu bajtova. Dakle, kod ovog tipa *socket-a*, osigurava se da će podaci doći do odredišta u redoslijedu u kojemu su poslani. Ovaj tip *socket* komunikacije je korišten u izradi ovog rada. „Datagram Sockets“ vrsta *socket*-a koristi dvosmjerni tijek komunikacije, a poruke se mogu primati drugačijim redoslijedom od načina slanja, te se također mogu primati duple poruke. Dakle, isporuka poruka nije osigurana, kao ni redoslijed tih poruka [15].

3.4. RobotStudio

RobotStudio je softverski alat kojega je razvila tvrtka ABB. RobotStudio se koristi u automobilskoj industriji, elektronici te općoj proizvodnji za zadatke poput zavarivanja, bojanja, montaže, rukovanja materijalima i slično. Također se koristi i u edukacijske svrhe. RobotStudio nudi niz virtualnih kontrolera koji se temelje na stvarnim kontrolerima robota, te su točna kopija stvarnih kontrolera. Ovo omogućuje simuliranje visoke točnosti te preuzimanje stanja kontrolera na stvarne robote. Jedna od ključnih značajki RobotStudio-a je mogućnost programiranja izvan mreže (eng. offline). Ovime je moguće testirati rad robota u virtualnom okruženju, smanjujući troškove zastoja rada robota. RobotStudio sadrži integrirane biblioteke modela stvarnih robota te raznih robotskih alata i objekata za stvaranje virtualnog okruženja. Također omogućuje integraciju CAD modela. Bitna značajka RobotStudi-a je RAPID programski jezik koji nudi tekstualno sučelje gdje se mogu pisati programi i skripte za kontrolu kretanja robota ali i interakciju sa drugim sustavima [16].

4. Izrada programa i simulacije u Python-u i RobotStudio -u

U ovom dijelu rada opisuje se cijeli proces stvaranja programa koji detektira objekt na kameri i informaciju šalje u RobotStudio. Zatim na temelju dobivene informacije robot prima pokazani predmet i premješta ga na mjesto predviđeno za točno taj predmet. U isto vrijeme, ako se osoba pojavi na kameri, robot stane. Zbog nemogućnosti *socket* komunikacije kontrolera robota koji se nalazi u laboratoriju na fakultetu, još je bilo potrebno napraviti simulaciju i modelirati okruženje u RobotStudio programu. Sav korišteni kod nalazi se u prilogu. Ovaj dio završnog rada može se podijeliti na četiri glavna dijela:

1. Treniranje modela neuronske mreže za detekciju objekta,
2. Komunikacija između dva programa; Python i RobotStudio,
3. Programiranje kretanja robota u RobotStudio programu,
4. Stvaranje simulacije i okruženja u RobotStudio programu.

4.1. Klasifikacija slika i priprema za treniranje

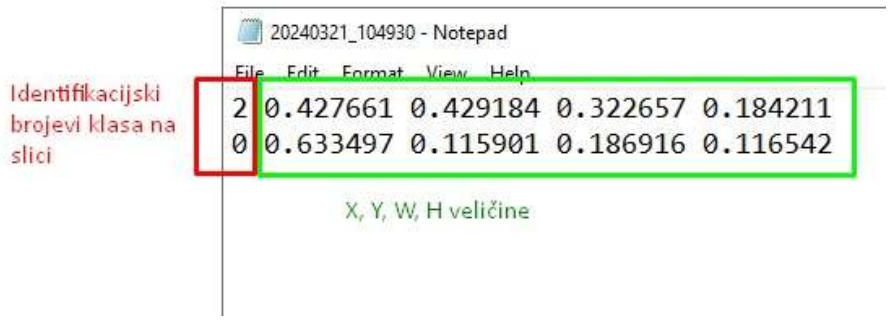
Prije treninga modela potrebno je skupiti podatke na temelju kojih se model trenira. Za YOLO to se odnosi na slike i *.txt datoteke. Potrebno je da model na kameri detektira osobu, kocku i valjak. Dakle, slike koje se koriste u treningu sadrže te objekte. Svaki objekt, odnosno klasa, koja se detektira ima određeni identifikacijski broj. U ovom slučaju za klasu osobe to je broj 0, za klasu kocke 80 a valjka 81. Zašto baš ti brojevi biti će objašnjeno u nastavku gdje se klase definiraju. Za sada je bitno samo da svaka klasa ima vlastiti identifikacijski broj. Potrebno je uslikati veliku količinu slika predmeta koji će biti detektirani. Zatim se za svaki objekt na slikama stvara okvir i daje im se broj klase (slika 4.1).

Slika 4.1 – Stvaranje okvira oko objekta i dodavanje klase

Za taj proces postoji velika količina web stranica i programa. Zatim je na istim stranicama i programima moguće izvesti te slike. U isto vrijeme se izvedu i slike i *.txt datoteke koje imaju isti naziv kao njima pripadajuće slike. Otvaranjem izvezene slike može se vidjeti da oko nje ne postoji okvir niti identifikacijski broj klase (slika 4.2.).

Slika 4.2. – Primjer slike korištene za trening

Ako se otvori .txt datoteka istog imena kao slika 5.2, vidljivo je da se u njoj nalaze brojevi (slika 4.3.).



Slika 4.3. – Brojevi unutar .txt datoteke

Prvi broj svakog retka označava o kojoj klasi je riječ. Znamo da je broj klase kocke 80, stoga ostatak brojeva tog retka odnosi se na klasu kocke. Slijedeća dva broja označavaju koordinate središta okvira korištenog za klasu kocke. Posljednja dva broja označavaju širinu i visinu okvira. Ovaj format se zove „xywh“ format, gdje w označava širinu (eng. width), a h visinu (eng. height). Ovaj format ne uzima broj piksela slike. Iz tog razloga niti jedan broj nije veći od 1. Gornji lijevi kut slike ima koordinate 0,0, a donji desni 1,1 (slika 4.4.).

Slika 4.4. Primjer xy koordinatnog sustava xywh formata

U treningu je korišteno 1583 slika, od kojih je 267 mojih. Ostalih 1316 slika, zajedno sa odgovarajućim txt datotekama, je pomoću Python skripte preuzeto sa interneta. To su neke od slika klase „person“ korištene za pred trening u COCO skupu podataka. Više o tome je pojašnjeno u nastavku. Za slike i txt datoteke se stvaraju 6 mapa. Prvo se stvore mape pod nazivima „train“ i „val“ (skraćeno od „training“ i „validation“). U svakoj od njih se stvore po dvije mape pod nazivima „images“ i „labels“. U mapu images se ubacuju slike, a u labels *.txt datoteke slika ubačenih u mapu images. U mapu train je ubačeno 1509 slika, a u mapu val 344 slike. Ovime je model neuronske mreže spreman za treniranje.

4.2. Treniranje modela neuronske mreže YOLOv8

Prvi dio izrade programa odnosi se na treniranje YOLO modela neuronske mreže. Za to je potrebno instalirati određene pakete i biblioteke; ultralytics i PyTorch. Zatim se stvara Python program, nazvan train.py, u koji se učitava YOLO klasa ultralytics biblioteke.

```
from ultralytics import YOLO
```

Slijedi definiranje modela neuronske mreže te definiranje podataka koji će biti korišteni za treniranje i parametara koji određuju način treniranja. Prvo se definira da je korištena srednja verzija YOLO modela neuronske mreže, yolov8m, koji je pred treniran na COCO skupu podataka. Srednja verzija modela definirana je sufiksom „m“. Postoje još i „n“, što označava najmanju verziju (nano), mala verzija „s“(eng. small), velika verzija „l“(eng. large) te najveća verzija x (eng. extra large - XL). Najmanja verzija se najbrže trenira i najbrže detektira, no točnost je također najmanja. Za trening najveće verzije je potrebno najviše vremena te je brzine detekcije najduža, ali točnost je najveća. Srednja verzija je odabrana zbog dobrog omjera između brzine detekcije i točnosti. O pred treniranom modelu mreže i COCO skupu podataka će nešto više biti objašnjeno u nastavku. Parametar „epochs“ određuje broj epoha, odnosno koliko se puta prođe kroz podatke tijekom treninga. Ovaj parametar ne bi trebao biti pre velik jer model može postati jako dobar u detektiranju treniranih podataka, ali lošiji u detektiranju podataka sa kojima se nije susreo. Zadana vrijednost im je 100, što je promijenjeno na 20. Parametar „workers“ određuje broj odvojenih procesa koji se odvijaju tijekom treniranja. Zadana vrijednost ovog parametra je 8, ali zbog manjka radne memorije računala bilo je potrebno smanjiti ju na 1. Ovime se ujedno i usporio proces treniranja, ali u protivnom dolazi do rušenja procesa. Ovdje su promijenjena samo dva parametra. To ne znači da drugi parametri ne postoje, već pošto nisu promijenjeni, njihova zadana vrijednost se ne mijenja. Parametar data je popraćen „string-om“ koji određuje točnu lokaciju „data.yml“ datoteke. Ona definira mjesto na disku gdje se nalaze mape imena „train“ i „val“. Isto tako definira broj i imena klasa treniranog modela.

```
model = YOLO('yolov8m.pt')

if __name__ == '__main__':
                                          results                    =
model.train(data='c://Users//pudin//runs//detect//Detection//data.yml
', epochs=20, workers=1)
```

Sadržaj data.yml datoteke je napisan u kodu koji slijedi. Radi duljine koda nije prikazan kompletan kod već samo važniji dio. Prvo se definira mjesto na računalu gdje se nalaze datoteke „train“ i „val“. Zatim se definira broj klasa. Ispod „names“ parametra popisane su sve klase treniranog modela te se ispred njih nalazi njihov broj. Ovdje su važne samo 3 klase; *person*, kocka

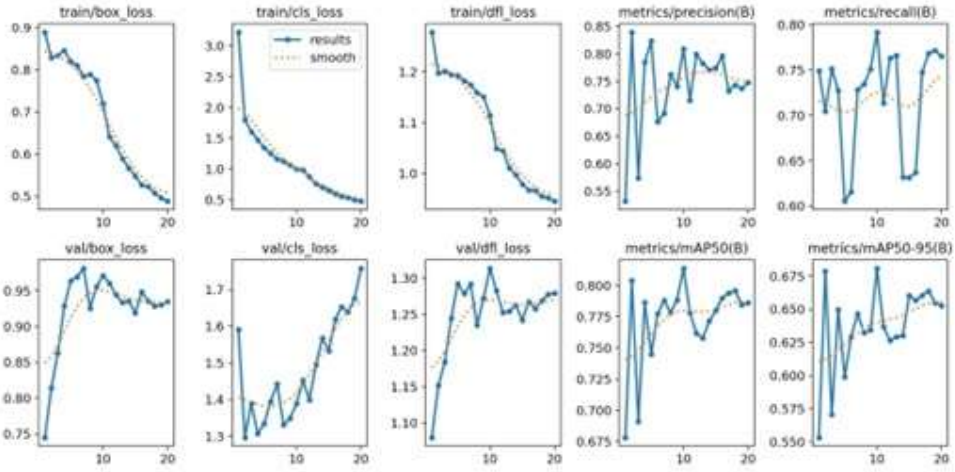
i valjak. Korišteni model neuronske mreže nije cijeli treniran iz nule, već se koristi pred trenirani YOLO model, koji je pred treniran na COCO skupu podataka. Zbog toga, on u sebi već sadrži 80 klasa, jedna od kojih je potrebna u ovom radu. To je klasa *person*. Ovo je važno jer ova klasa mora biti dobro trenirana da se na kameri sa što većom preciznosti detektira osoba te da se robot potom može zaustaviti. Važno je napomenuti da se klase koje se ne koriste u dodatnom treniranju neće detektirati i biti korištene. Čim su na 80 pred treniranih klasa dodane nove klase, pred trenirani model mreže više ne detektira one klase koje se ne koriste u novom treningu. Razlog tomu je pojam zvan kao „Catastrophic Forgetting“, gdje neuronska mreža zaboravlja sve ili većinu prethodno naučenih informacija kada se vježba na novim podacima. To se obično javlja kod sekvencijalnog obučavanja na različitim zadacima ili skupovima podataka, čime se prebrišu naučene „težine“ (eng. weights) iz ranijih zadataka. To se u ovom slučaju može iskoristiti pošto je potrebna samo jedna pred trenirana klasa (*person*) i dodaju se dvije nove (kocka i valjak). Dakle, sve ostale pred trenirane klase poput „bicycle“ , „car“, „toothbrush“ neće biti detektirane. No, korištenje stare klase nije tako jednostavno. Još uvijek se treba koristiti velika količina klasificiranih slika te klase kako model mreže ne bi „zaboravio“ podatke koje zna o toj klasi. Rezultat je model neuronske mreže koji puno bolje detektira klasu *person* nego model mreže koji bi bio treniran „iz nule“ sa istim tim slikama i istom količinom treninga. Klase kocka i valjak dodane su kao nastavak pred treniranih 80 klasa, na mjesta 80 i 81. Skripta se pokrene i kada se trening završi stvori se mapa „train2“. U njoj se nalaze svi rezultati treninga poput onih vidljivih na slici 4.5, argumenti korišteni u treningu te „težine“ koji će se koristiti u programu za detektiranje.

```
path: 'c:\Programiranje\ML\COCO'
train: train
val: val

nc: 82

names:
  0: person
  1: bicycle
  2: car
  3: motorcycle
```

  …

```
  79: toothbrush
  80: kocka
  81: valjak
```

Rezultati treninga (slika 4.5) ne izgledaju idealno, ali kasnije isprobavanje modela mreže će pokazati da je istrenirani model mreže dovoljno dobar.



Slika 4.5. – Rezultati treninga

4.3. Program za detektiranje

U nastavku će biti objašnjeno kako napraviti program koji detektira objekte na definiranoj kameri. Zatim te informacije šalje programu server.py. No prije toga, potrebno je isprobati istrenirani model neuronske mreže. U slučaju da istrenirani model mreže ne daje očekivane rezultate, potrebno ga je opet istrenirati.

4.3.1. Isprobavanje modela neuronske mreže

Za isprobavanje istreniranog modela neuronske mreže potrebno je uvesti YOLO biblioteku. Zatim se definira lokacija i ime težine modela mreže koja će biti korištena za detektiranje objekata. Na kraju se pokreće funkcija predict() sa definiranim modelom mreže. Parametri funkcije će biti detaljnije opisani u nastavku.

```
from ultralytics import YOLO

model = YOLO(r'train2\weights\last.pt')

results = model.predict(source= "1", show=True)
```

Pokretanjem programa se dobiju rezultati na slici 4.6. Pošto su rezultati zadovoljavajući, model mreže se može koristiti za daljnje programiranje.

Slika 4.6.- Rezultat probnog pokretanja modela

4.3.2. Stvaranje programa za detektiranje objekta

Slijedeći korak je pisanje programa za detektiranje objekta koji se nalazi na kameri. Prvo se uvoze potrebne biblioteke YOLO i cv2. YOLO je potreban za detektiranje objekta i korištenje treniranog modela mreže, a cv2 za kameru i manipuliranjem kadra, to jest slike (eng. frame) kamere. Iz server programa, kojeg još nisam opisao, uvozi se funkcije *update*. Više o toj funkciji biti će objašnjeno kasnije.

```
from ultralytics import YOLO
from server import update
import cv2
```

Definira se kamera. Broj na kraju označava na koji se ulaz računala odnosi. Na primjer, ako računalo ima dvije kamere, onda je jedna definirana brojem 0, a druga 1.

```
cam = cv2.VideoCapture(1)
```

Težine modela mreže korištene u detektiranju su definirane ovdje. To je datoteka zvana last.pt koja se nalazi u mapi sa rezultatima treninga.

```
model = YOLO(r'train2\weights\last.pt')
```

Stvara se lista *obj_list* koja će biti potrebna kasnije. U njoj će se nalaziti imena detektiranih objekata.

```
obj_list = []
```

Definira se funkcija *predict()*. U njoj se koriste prijašnje definirane varijable *model*, *cam* i *obj_list*. Ispred njih se nalazi prefiks global. Da njega nema, ovo bi bile novonastale varijable koje vrijede samo za funkciju *predict()*. Dakle, pošto su varijable definirane izvan funkcije, potrebno je dodati prefiks global.

```
def predict():
    global model
    global cam
    global obj_list
```

Stvara se beskonačna petlja. U njoj se definiraju varijable *frame* i *results* te se lista *obj_frame* isprazni. Varijabla *frame* označava okvir kamere, odnosno svaku sliku od koje je video kamere *cam* sastavljen. Funkcijom *read()* se čita slika kamere. Zatim se ta ista slika koristi kao parametar funkcije *predict()* u varijabli *results*. Ostali parametri funkcije *predict()* čije se vrijednosti mijenjaju su show, *iou* i *conf*. Parametar *iou* je smanjen sa 0.7 na 0.5 kako bi se smanjio broj okvira koji se preklapaju. *Conf* je povećan sa 0.25. na 0.5 kako se objekt ne bi detektirao ako povjerenje (eng. confidence) nije barem 0.5. Varijabla *show* je upaljena da se vide nazivi objekata na slici. Funkcija *predict()* na temelju ovih parametara i modela detektira objekte koje se nalaze na kameri. Varijabla *results* je definirana kao rezultat funkcije *predict()*. U njoj se nalazi lista rezultata poput „Boxes“ u kojoj se nalazi ime detektiranog objekta. Slijedećim koracima opisano je kako doći do tih imena.

```
while True:
         _,frame = cam.read()
         results  =  model.predict(source=frame,  show=True,  iou=0.5,
conf=0.3)[0]
        obj_frame = []
```

Pomoću for petlje se prolazi kroz svaki rezultat u varijabli *results*. Unutar tih rezultata je i jedan pod nazivom „boxes“.

```
        for r in results:
            for b in r.boxes:
```

Zatim se i kroz njegov svaki podatak prolazi kako bi se došlo do podatka „cls“. *Cls* je podatak koji daje identifikacijski broj detektirane klase. Potrebno ga je iz „tensor“ tipa podatka pretvoriti u „int“, odnosno cijeli broj. Varijabla *obj_index* poprima oblik tog broja. Pomoću *names()* funkcije se može dobiti ime klase za definirani model mreže. Kao parametar je stavljen identifikacijski broj klase u obliku cijelog broja. Varijabla *obj_name* poprima oblik dobivenog imena. Dobiveni broj klase i ime klase se zajedno printaju. Ime se zatim stavi u listu *obj_frame*.

```
                obj_index = int(b.cls)
                obj_name = model.names[int(b.cls)]
                print(obj_index,obj_name)
                obj_frame.append(obj_name)
```

Lista *obj_list* poprima podatke liste *obj_frame*. Na kraju se poziva funkcija *update*, a lista *obj_list* joj je parametar. Ovo je funkcija server.py programa o kojemu će više biti napisano kasnije.

```
        obj_list = obj_frame
        update(obj_list)
```

4.4. Program servera i slanje podataka u RobotStudio

Komunikacije između Python programa i RobotStudio programa izvedena je pomoću mrežne *socket* komunikacije. Biblioteku *socket* je potrebno uvesti. Još se uvode biblioteke *threading*, *time* i *sys*. „Time“ biblioteka omogućava čekanje programa na određeno vrijeme funkcijom *sleep()*. Biblioteka *threading* omogućuje izvedbu više funkcija programa istovremeno. Biti će potrebne 4 niti (eng. thread) koje će se koristiti za slijedeće zadatke:

- korisničko sučelje,
- rad kamere i stalno detektiranje objekata na njoj,
- funkcija za slanje podatka u RobotStudio ako se na kameri nalazi čovjek u svrhu zaustavljanja kretanja robota,
- funkcija automatskog slanja podataka u RobotStudio nakon četiri sekunde prisutnosti objekta na kameri.

Svaka od ovih niti biti će detaljnije objašnjena u nastavku. Biblioteka *sys* pomoću funkcije *exit()* omogućava izlaz iz niti.

```
import socket
import threading
import time
import sys
```

Definira se IP adresa i *port*. Pošto se RobotStudio i Python koriste na istoj mreži, koristi se lokalna IP adresa.

```
host = '127.0.0.1'
port = 1025
```

Definira se server i pridodaju mu se IP adresa i *port*. Ovom naredbom definira se korištenje IPv4 IP protokola i TCP protokola. Na kraju se omogućuje „slušanje“ servera, odnosno prihvaćanje veza na server.

```
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind((host,port))
server.listen()
```

Stvara se lista zvana *clients* u koju će se dodavati ime klijenta iz RobotStudio programa. Varijabla *received_data* će kasnije poprimiti oblik podataka koji se pošalju iz RobotStudio-a. U listu *updated_objects* će se stavljati objekti koji su detektirani na kameri.

```
clients = []
received_data = ''
updated_objects = []
```

Funkcija *update()* je spomenuta ranije u dijelu koji opisuje detektiranje objekata. Ona se poziva iz programa prediction.py. Funkcija *update()* u listu *updated_objects* stavlja imena svih objekata koji se nalaze na slici kamere u tom trenutku. Pošto se pozove za svaku novu sliku na kameri, lista *updated_objects* se osvježava onoliko puta u sekundi koliko kamera ima FPS-a (eng. Frame Per Second).

```
def update(objects):
    global updated_objects
    updated_objects= objects
```

Funkcija *receive()* prihvaća samo jednog klijenta. Ispisuje njegovu IP adresu te ga stavlja u listu klijenata. Zatim stvara nit u kojoj se pokreće funkcija *handle()* sa klijentom kao argumentom funkcije. Nakon toga stvara još jednu nit, u kojoj se pokreće funkcija *define_obj()*. Kako ove funkcije rade biti će opisano u nastavku. Funkcija *receive()* je isto pokrenuta u niti. Ona se gasi funkcijom *exit()* pošto više nije potrebna.

```
def receive():
    print('Server is running...')
    print("recieving...")
    global clients
    while len(clients) == 0:
        client, adress = server.accept()
        print(f'Connected with {str(adress)}')
        clients.append(client)
        thread1 = threading.Thread(target=handle, args=(client,))
        thread4 = threading.Thread(target=define_obj)
        thread1.start()
        thread4.start()
        sys.exit()
```

Funkcija *handle()* služi za zaustavljanje robota kada se na kameri pojavi osoba. Kao parametar uzima klijenta s kojim onda komunicira. Stalno provjerava podatke varijable *received_data*. Kada *received_data* ima podatak „Check“ u sebi, funkcija *handle()* provjeri listu u kojoj se nalaze objekti na kameri. Ako je jedan od tih objekata osoba, funkcija *handle()* će u RobotStudio poslati podatak „alarm“, što dovodi do zaustavljanja robota. Funkcija *handle()* nastavlja raditi nakon toga na isti način. Nakon što se osoba makne sa kamere, u RobotStudio se šalje podatak „none“ što omogućava daljnji rad robota. Svi podaci koji se šalju i primaju se šifriraju i dešifriraju u ASCII formatu.

```
def handle(client):
    global updated_objects
    while True:
        global received_data
        received_data = client.recv(1024).decode('ascii')
        if received_data == "Check":
            if "person" in updated_objects:
                clients[0].send("alarm".encode('ascii'))
                print("sent...")
            else:
                clients[0].send("none".encode('ascii'))
                print("sent...")
        else:
            print(received_data)
```

Funkcija *define_obj()* služi za automatsko detektiranje objekta. Na temelju podatka u varijabli *received_data*, ova funkcija detektira u kojoj fazi rada se nalazi robot. Ako robot ne radi i čeka informaciju koji objekt treba prihvatiti, iz RobotStudio-a se pošalje *string* "Client: Waiting for object recognition...". Varijabla *received_data* poprimi oblik tog podatka te se zatim pokreće funkcija *define_obj()*. Stvara se varijabla *sent* i poprima oblik logičke nule. Dok je u nuli, odnosno dok je netočna, pokreće se *while* petlja. U njoj se provjerava je li u kadru kocka ili valjak. Samo jedan objekt smije biti u kadru. Kada se taj uvjet ispuni, poziva se čekanje programa od četiri sekunde. Ako se isti objekt još uvijek nalazi na slici kamere, pokreće se petlja *det_obj*, a varijabla *sent* se postavlja u logičku jedinicu kako bi se *while* petlja prekinula. Ako se za vrijeme čekanja od četiri sekunde predmet pomakne, *while* petlja se nastavlja ponavljati sve dok se predmet opet ne detektira.

```
def define_obj():
    global updated_objects
    while True:
        global received_data
        print(f'received data is {received_data}')
       if received_data == "Client: Waiting for object recognition..."
or received_data == "Client: received" :
            sent = False
            while sent == False:
                    if ("kocka" in updated_objects or "valjak" in
updated_objects) and len(updated_objects)==1:
                   print(f'{updated_objects[0]} has been found in the
frame!')
                    time.sleep(4)
                       if ("kocka" in updated_objects or "valjak" in
updated_objects) and len(updated_objects)==1:
                           sent = True
                           det_obj()
                    else:
                           print("Object has been moved.")
                else:
                    print("No objects detected...")
```

Funkcija *det_obj()* služi za slanje informacije o kojem se predmetu na kameri radi. Čita podatke liste *updated_objects*. Ako se više podataka u njoj nalazi, ništa se ne šalje u RobotStudio. Ako se u listi nalazi samo jedan objekt, ime tog objekta šalje u RobotStudio.

```
def det_obj():
    global updated_objects
    print(updated_objects)
    if len(updated_objects)>1:
        print("Multiple objects detected!")
    elif len(updated_objects) == 0:
        print("No objects in frame.")
    elif updated_objects[0] == "kocka":
        clients[0].send("kocka".encode('ascii'))
    elif updated_objects[0] == "valjak":
        clients[0].send("valjak".encode('ascii'))
```

4.5. Programiranje korisničkog sučelja

Korisničko sučelje je napravljeno u svrhu lakšeg pokretanja svih programa napravljenih u Pythonu. U njemu se nalaze dva gumba. Jedan pokreće *server.py* program, a drugi *prediction.py* program. Korisničko sučelje je napravljeno pomoću alata PyQt6. Prvi korak je bio uvoženje biblioteka. Iz PyQt6 „Widgets“ biblioteke uvezeni su alati koji će biti potrebni pri stvaranju korisničkog sučelja. Uvezeni su *server.py* i *prediction.py* programi jer će se pokretati iz ovog korisničkog sučelja. *Threading* je uveden jer će se svaki od tih programa pokrenuti u zasebnoj niti.

```
from PyQt6.QtWidgets import QApplication, QWidget, QPushButton,
QVBoxLayout
import sys
import server
import prediction
import threading
```

Zatim se stvara klasa „MyApp“, te se definiraju parametri klase poput imena prozora, dimenzija sučelja te se definira prostor pomoću *QVBoxLayout()* klase.

```
class MyApp(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('Window')
        self.resize(900,700)
        layout = QVBoxLayout()
        self.setLayout(layout)
```

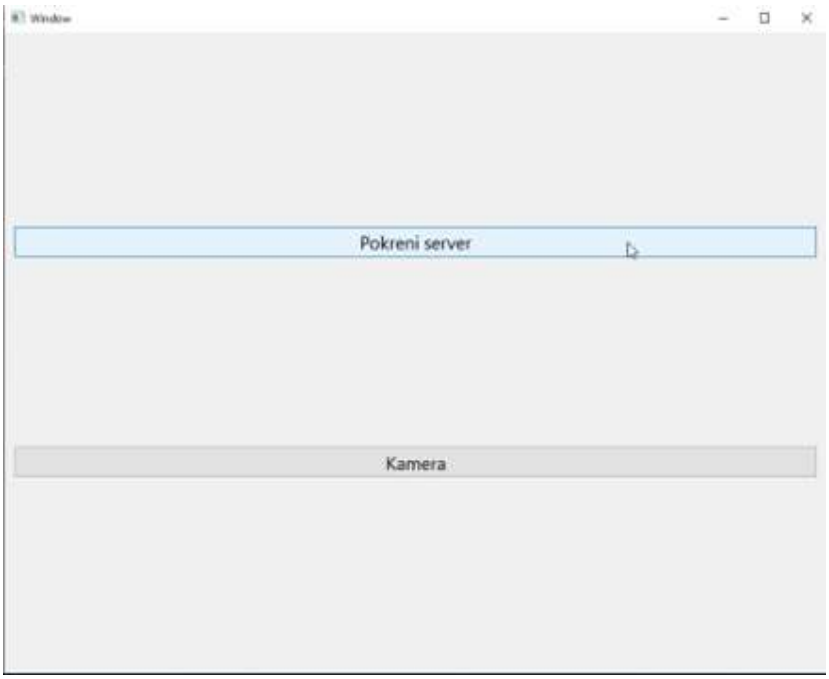
Potom se dodaju gumbovi te se svakom gumbu dodaje određena funkcija. Prvi gumb u novoj niti pokreće program server.py i njegovu funkciju *receive()*. Drugi gumb u novoj niti pokreće program prediction.py i njegovu funkciju *predict()*.

```
            server_start  =  QPushButton("Pokreni  server",clicked  =
thread3.start)
        kamera = QPushButton("Kamera", clicked = thread2.start)

        layout.addWidget(server_start)
        layout.addWidget(kamera)

thread2 = threading.Thread(target=prediction.predict)
thread3 = threading.Thread(target=server.receive)
```

Na kraju se mijenja veličina fonta aplikacije, te se aplikacija pokreće.

```
if __name__ == "__main__":
    app = QApplication(sys.argv)
    app.setStyleSheet('''
        QWidget {
                    font-size: 25px;
        }
        QPushButton {
                    font-size: 20px;
        }
                    ''')
    window = MyApp()
    window.show()
    app.exec()
```

Izgled korisničkog sučelja se može vidjeti na slici 4.7



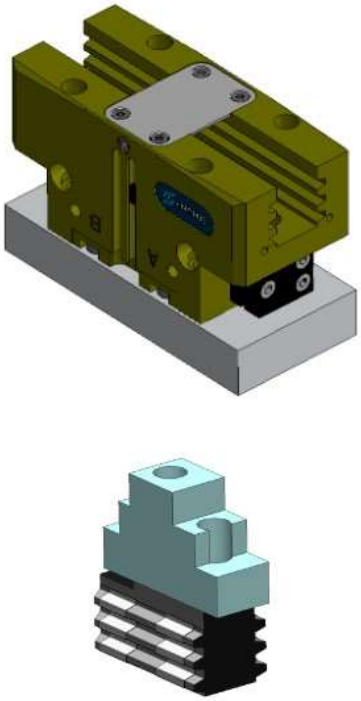
Slika 4.7. – Korisničko sučelje

4.6. Postavljanje okruženja u RobotStudio programu

Prije početka programiranja u RAPIDu, potrebno je napraviti osnovno okruženje. To podrazumijeva dodavanje robota, definiranje alata robota, stvaranje mehanizma alata i stvaranje predmeta kojima će robot upravljati.

4.6.1. Dodavanje modela stola i alata robota

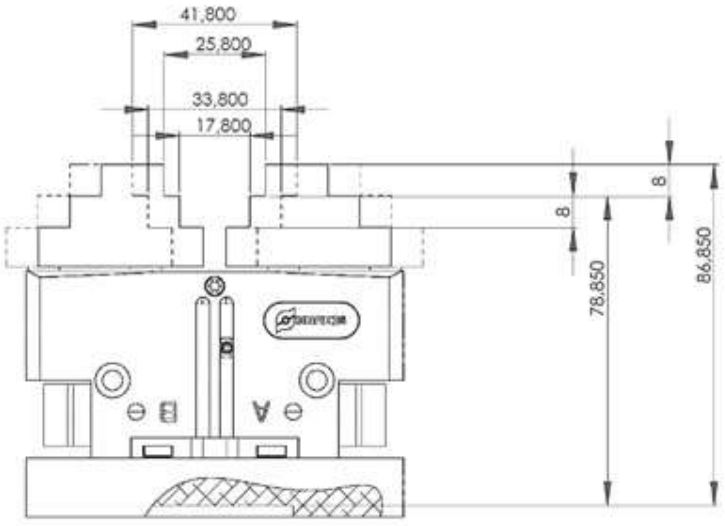
Robot se dodaje iz izbornika „ABB Library“ koji se nalazi u „Home“ kartici. Dodaje se robot IRB 120. Zatim se dodaje model stola naziva UNIN table. Taj je model preuzet sa Merlina te unesen sa „Import Geometry“ naredbom koja se nalazi u „Modeling“ kartici. Idući korak je unijeti alat robota. Modeli tijela i prstiju alata (slika 4.8) također su preuzeti sa Merlina. Uvoze se „Import Geometry“ naredbom.



Slika 4.8. – Tijelo i prst alata [17]

4.6.2. Definiranje alata i stvaranje mehanizma

Tijelo i prste alata je potrebno spojiti u jedni cjelinu. Dimenzije na temelju kojih su postavljene pozicije prstiju si prikazane su na slici 4.9.

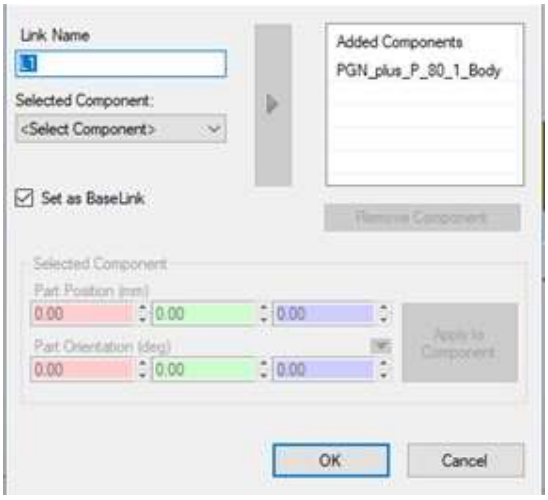


Slika 4.9. Udaljenosti između dva prsta i tijela alata [17]

Prsti se postavljaju u prikazanu poziciju naredbama „Set Position“ i „Rotate“. Udaljenosti se mogu provjeravati naredbom „Measure“ koja se nalazi u „Modeling“ kartici. Idući korak je stvaranje mehanizma alata. Cilj je napraviti tri pozicije prstiju:

1. Početna pozicija,
2. Krajnje otvorena pozicija,
3. Pozicija prihvata objekta.

Početna pozicija je ona koju prsti imaju prije pomaka. Prije prihvata predmeta se pomiču u krajnje otvorenu poziciju. Kako bi prihvatili predmet, prsti se kreću u poziciju prihvata objekta. Odabire se alat te se pokreće naredba „Create Mechanism“ u kartici „Modeling“. Odabrano ime mehanizma je *PGN_P80_1*. Kao tip mehanizma se odabire „Tool“. Zatim se definiraju veze (eng. links) mehanizma. Prvo se dodaje veza L1, kojoj se dodaje komponenta tijela (slika 4.10.). Označi se da je ovo „BaseLink“.



Slika 4.10. – Stvaranje poveznice L1

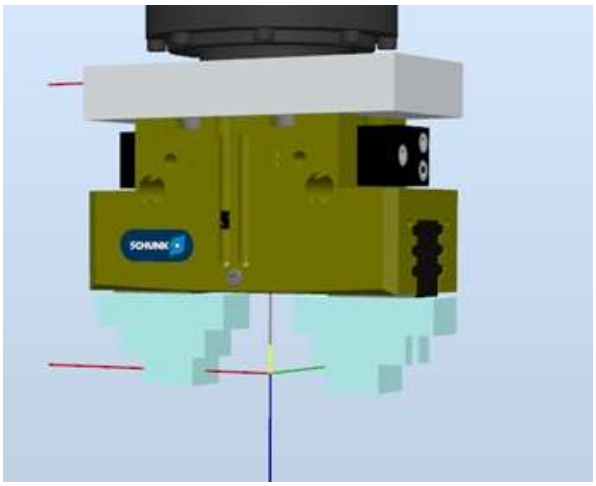
Prsti mehanizma se dodaju kao veze L2 i L3, s jedinom razlikom što se kod njih ne označi da su „BaseLink“. Zatim se dodaju zglobovi mehanizma (eng. Joints). Za zglob J1 se kao glavnu vezu (eng. Parent Link) uzima veza L1, a podređenu vezu (eng. Child Link) veza L2. Kao tip zgloba se odabire „Prismatic“, što znači da se radi o pravocrtnom gibanju. Definiraju se parametri pozicija, gdje prva pozicija ostaje u nuli. Drugoj poziciji se mijenja x parametar u -8.00mm. Minimalno ograničenje se definira kao 0mm, a maksimalno kao 8mm. Na slici 4.11 su prikazana svojstva zgloba J1.

Slika 4.11.- Svojstva zgloba J1.

Proces se ponovi za zglob J2, sa jedinim razlikama što se odabere veza L3, te se umjesto -8mm stavi 8mm kao x parametar druge pozicije. Idući korak je definiranje podataka alata. Odabrani su parametri kao na slici 4.12. Pozicija je pomaknuta za 83mm na osi z. Centar gravitacije je pomaknut za 40mm po osi z. Masa je postavljena na 1kg.

Slika 4.12.- Parametri podataka alata

Na taj način koordinatni sustav alata je točno između prstiju alata što se vidi na slici 4.13.



Slika 4.13.- TCP pozicija u odnosu na prste alata

Ovime su svi parametri mehanizma definirani i moguće je dodati pozicije mehanizma. Prva pozicija se naziva „HomePose“, te su njeni parametri zglobova u nuli. Druga pozicija je nazvana „OtvorenaPrihvatnica“, a njeni parametri zglobova su za svaki prst postavljeni na 8 mm. Treća pozicija je nazvana „otvori_Kocka/Valjak“, a njeni parametri zglobova su za svaki prst postavljeni na 4.6 mm. To znači da će otvor između prstiju biti 35mm. Prema tome će biti modelirani modeli kocke i valjka. Alat se pričvrsti na robota naredbom „Attach to“ koje se nalazi u izborniku koji se pojavi desnim klikom na alat.

4.6.3. Stvaranje signala za upravljanje alatom

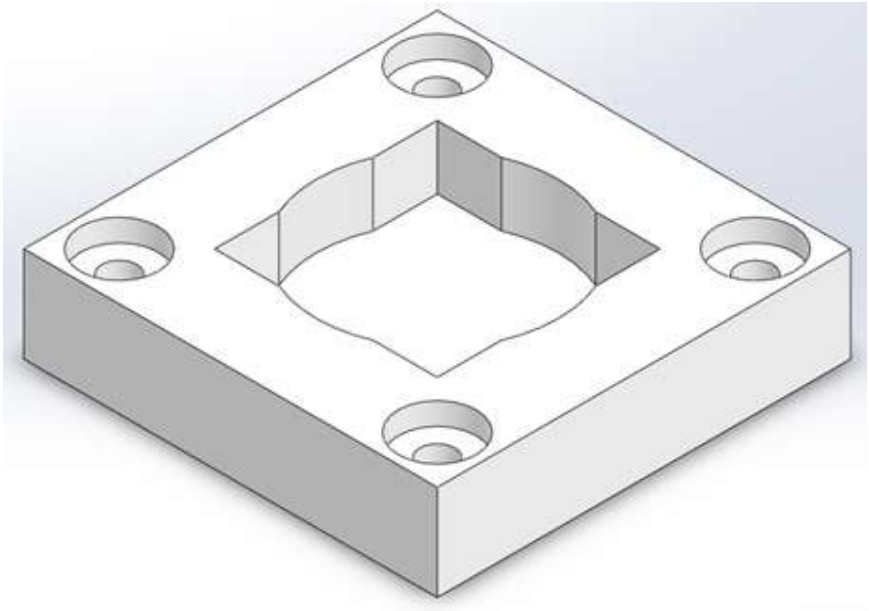
Stvoreni mehanizam ima tri pozicije. Kako bi došao u njih potrebno je stvoriti signale kojima se upravljaju pozicije mehanizma. Signali se stvaraju u kartici „Controller“, pritiskom na „Configuration“ izbornik te odabirom opcije „I/O Systems“. Zatim se odabere „Signal“ opcija. U ovom prozoru se desnim klikom pojavljuje opcija „New Signal“. Dodaju se dva digitalna izlaza: „doOtvoreno“ i „doPrihvatObjekta“. Razina pristupa kod oba signala je postavljena na „All“. Idući korak je ove signale spojiti sa pokretima mehanizma. Značajka „Event Manager“ koja se nalazi u „Simulation“ kartici to omogućava. Dodaju se četiri događaja (eng. events). Prva dva su vezana za signal „doOtvoreno“. Pritiskom na opciju „Add“ se započinje sa dodavanjem događaja. Uvjet aktivacije signala je postavljen na „On“. Vrsta događaja je „I/O signals changed“. Izabire se signal „doOtvoreno“. Kao izvor signala se odabire aktivni kontroler. Pod „Trigger Condition“ opcijom se odabire „Signal is true („1“)“. Zatim se definira događaj. Odabire se „Move Mechanism to Pose“. Kao poziciju u koju želimo da se mehanizam pomakne se odabire „OtvorenaPrihvatnica“. Ovaj proces se ponovi za signal „doOtvoreno“, ali se kao „Trigger Parameter“ odabere 0, a završna pozicija „HomePosition“. Dakle, ako se signal „doOtvoreno“ upali, prsti alata će se pomaknuti u poziciju „OtvorenaPrihvatnica“. Kada se signal „doOtvoreno“ ugasi, prsti alata se vraćaju u poziciju „HomePosition“. Proces dodavanja signala se ponovi za signal „doPrihvatObjekta“. Mehanizam se dovodi u poziciju „otvori_Kocka/Valjak“ kada se signal“ doPrihvatObjekta“ upali. Kada se isti signal ugasi, mehanizam se dovodi u poziciju „OtvorenaPrihvatnica“. Dodani događaji su prikazani slikom 4.14.

Activation	Trigger Type	Trigger System	Trigger Name	Trigger Parameter	Action Type	Action System	Action Name	Action Parameter
On	I/O	IRB_120_3kg_...	doOtvoreno	1	Move Mechanism to Pose		Move Mechanism to Pose	PGN_P80_1 : OtvorenaPrihvatnica
On	I/O	IRB_120_3kg_...	doOtvoreno	0	Move Mechanism to Pose		Move Mechanism to Pose	PGN_P80_1 : HomePostion
On	I/O	IRB_120_3kg_...	doPrihvatObjekta	1	Move Mechanism to Pose		Move Mechanism to Pose	PGN_P80_1 : otvori_Kocka/Valjak
On	I/O	IRB_120_3kg_...	doPrihvatObjekta	0	Move Mechanism to Pose		Move Mechanism to Pose	PGN_P80_1 : OtvorenaPrihvatnica

Slika 4.14.- Dodani događaji

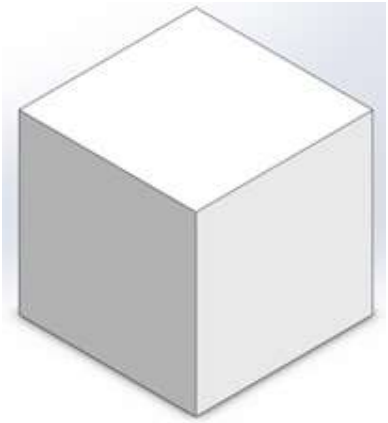
4.6.4. Modeliranje postolja, kocke i valjka

Modeli potrebni za simulaciju u RobotStudio-u so izrađeni u SolidWorks programu. Prvo je modelirano postolje koje je prikazano na slici 4.15. Na sebi sadrži četiri rupe M5 navoja. U sredini postolja nalazi se mjesto koje pričvršćuje kocku i valjak.



Slika 4.15. – Postolje za kocku i valjak

Model kocke napravljen je da bude dimenzija 35 mm x 35 mm. Na taj način dobro paše u postolje. Isto tako, ovo je razmak između prstiju robota kada zahvaćaju objekt. Model kocke prikazan je na slici 4.16.



Slika 4.16.- Model kocke

Model valjka ima dijametar 38mm. Ovime dobro paše na postolje, ali ne i za prihvat robotskim alatom. Pošto je zaobljen, postojeći alat ga ne može dobro prihvatiti. Iz tog razloga, na valjku su izrađena mjesta za zahvat prstima robotskog alata. Razmak svake strane tih mjesta je 35 mm što također pristaje razmaku prstiju robotskog alata za prihvat kocke. Model valjka prikazan je na slici 4.17.

Slika 4.17. – Model valjka

4.7. Stvaranje pametnih komponenti i logike stanice

Pametnim komponentama (eng. Smart Component) objektima u RobotStudio-u se dodaje funkcija. Ovime se omogućuje stvaranje simulacije stvarnog ponašanja robota. Pametna komponenta se stvara odabirom „Smart Component“ značajke koja se nalazi u „Modeling“ kartici.

4.7.1. Pametna komponenta postolja

Za stvaranje ove pametne komponente potreban je model postolja. Postolje modelirano u SolidWorks-u se izvozi kao .SAT datoteka. Zatim se uvozi u RobotStudio. Postavlja se na poziciju x 400mm, y -317.5mm, z 0mm. Zatim se odabere značajka „Smart Component“ i model postolja se povuče unutra. Ova pametna komponenta se preimenuje u „SC_Postolje“. Svrha ove pametne komponente je stvaranje modela kocke i valjka točno na mjestu postolja koje je predviđeno za njih. Pametnoj komponenti „SC_Postolje“ se u „Design“ kartici, klikom na „Inputs“ dodaju dva ulazna signala; „diKocka“ i „diValjak“. Klikom na „Outputs“ se dodaje izlazni signal „doPostavljeno“. U pametnu komponentu „SC_Postolje“ se dodaju unutarnje komponente (eng.

Child components) izvora (eng. source) klikom na „Add component“ naredbu u „Compose“ kartici. Unutarnje komponente izvora se preimenuju u „Source_Kocka“ i „Source_Valjak“. Funkcija komponenata izvora je stvaranje kopije odabranog modela na odabranim koordinatama. Dakle, prvo je potrebno uvesti modele koje se želi kopirati. Modeli kocke i valjka se iz SolidWorks-a izvoze kao .SAT datoteke, te se ubacuju u RobotStudio. Unutarnjoj komponenti „Source_Kocka“ se dodaje model kocke, a komponenti „Source_Valjak“ se dodaje model valjka. Zatim se u unutarnju komponentu „Source_Kocka“ (slika 4.18) upisuje pozicija x 400 mm, y -317,5 mm, z 43mm. Opcija „Transient“ se označuje što znači da će stvorena kocka biti privremena. U „PhysicsBehavior“ opciji se označuje „Kinematic“ što omogućuje da robotski alat premjesti novonastalu kocku tijekom simulacije. Ovaj se postupak ponovi za unutarnju komponentu „Source_Valjak“. Valjku se u z poziciju upisuje 5mm, a u polje orijentacije se stavlja 90 stupnjeva oko x osi.

Slika 4.18. Parametri komponente „Source_Kocka“

Komponente se sa ulazima i izlazom spajaju u funkcionalnu cjelinu u „Design“ kartici. Ulaz „diKocka“ se spaja sa „Execute“ signalom unutarnje komponente „Source_Kocka“. Signal „Executed“ unutarnje komponente „Source_Kocka“ se spaja sa digitalnim izlazom „doPostavljeno“. Ulaz „diValjak“ se spaja sa signalom „Execute“ unutarnje komponente „Source_Valjak“, a signal „Executed“ sa digitalnim izlazom „doPostavljeno“. Izgled spajanja prikazan je na slici 4.19. Dakle, pametna komponenta „SC_Postolje“ sadrži dva ulaza; „diKocka“ i „diValjak“, te jedan izlaz; „diPostavljeno“. Kada se ulaz „diKocka“ postavi u logičku jedinicu, unutarnja komponenta „Source_Kocka“ kopira model kocke na zadanu poziciju. Zatim postavi izlaz pametne komponente „SC_Postolje“ u logičku jedinicu.

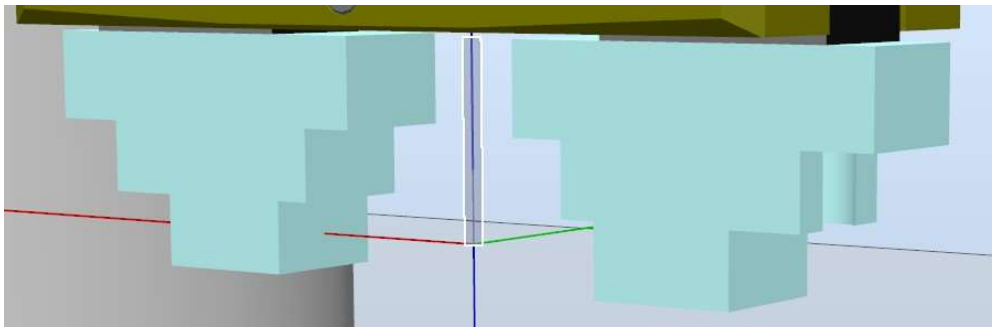
Slika 4.19. – Spojene unutarnje komponente „SC_Postolje“ pametne komponente

4.7.2. Pametna komponenta prihvatnice

Pametna komponenta prihvatnice služi za premještanje modela kocke i valjka sa postolja. Model robotskog alata se ubacuje u „Smart Component“ značajku. Pametnu komponentu se preimenuje u SC_Prihvatnica te se ubace slijedeće unutarnje komponente:

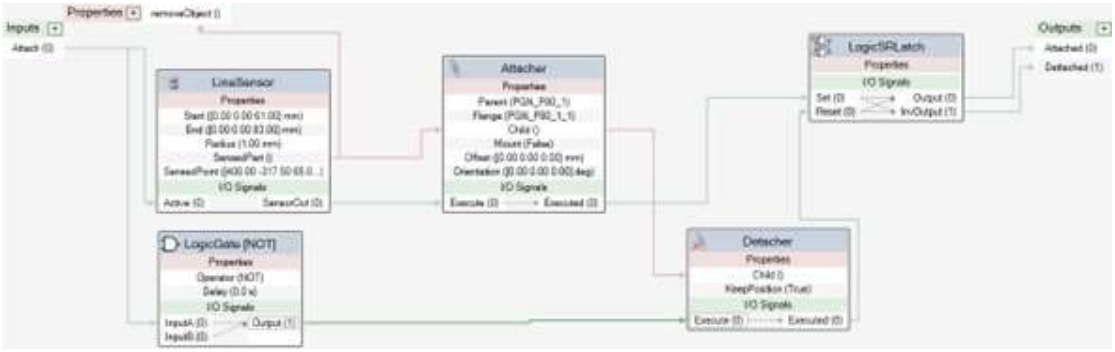
- „Attacher“ – Služi za spajanje određenog objekta sa alatom,
- „Detacher“ – Služi za odvajanje određenog objekta od alata,
- „LineSensor“ – Služi za detektiranje objekta,
- „LogicGate [NOT] – Služi za operaciju negiranja u unutarnjoj logici pametne komponente,
- „LogicSRLatch“ – Služi za „Set“ i „Reset“ operaciju u unutarnjoj logici pametne komponente.

Unutarnjoj komponenti senzora „LineSensor“ se početne koordinate postave između prstiju alata. Krajnje koordinate se postave da budu iste kao početne, ali z komponenta za 20 mm kraća. Radijus senzora se postavi na 1mm. Pozicija i izgled senzora su prikazani na slici 4.20.



Slika 4.20. – Izgled i pozicija unutarnje komponente LineSensor

U kartici „Design“ se doda ulazni signal „Attach“, te izlazni signali „Attached“ i „Dettached“. Zatim se dodaje svojstvo pametne komponente klikom na „Properties“. Svojstvo se imenuje „removeObject“. Ulazni signal „Attach“ se spaja sa signalom „Active“ senzora. Izlaz „SensorOut“ senzora se spaja sa ulaznim signalom „Execute“ unutarnje komponente „Attacher“. Svojstvo „SensedPart“ senzora se spaja sa „Child“ svojstvom unutarnje komponente „Attacher“. Svojstvo „SensedPart“ senzora se još i spaja sa svojstvom „removeObject“ pametne komponente. Izlazni signal „Executed“ unutarnje komponente „Attacher“ se spaja ulazom „Set“ unutarnje komponente „LogicSRLatch“. Svojstvo „Child“ unutarnje komponente „Attacher“ se spaja sa svojstvom „Child“ unutarnje komponente „Detacher“. Izlazni signal „Executed“ unutarnje komponente „Detacher“ se spaja sa ulazom „Reset“ unutarnje komponente „LogicSRLatch“. Izlaz „Output“ unutarnje komponente „LogicSRLatch“ se spaja sa izlazom „Attached“ pametne komponente „SC_Prihvatnica“. Izlaz „InvOutput“ unutarnje komponente „LogicSRLatch“ se spaja sa izlazom „Dettached“ pametne komponente „SC_Prihvatnica“. Ulaz „Attach“ pametne komponente „SC_Prihvatnica“ se spaja na ulazni signal „InputA“ unutarnje komponente „LogicGate [NOT]“. Izlaz „Output“ unutarnje komponente LogicGate [NOT] se spaja sa ulazom „Execute“ unutarnje komponente „Detacher“. Izgled veza između ulaza, izlaza i svojstva pametne komponente „SC_Prihvatnica“ sa unutarnje komponentama je prikazan na slici 4.21.

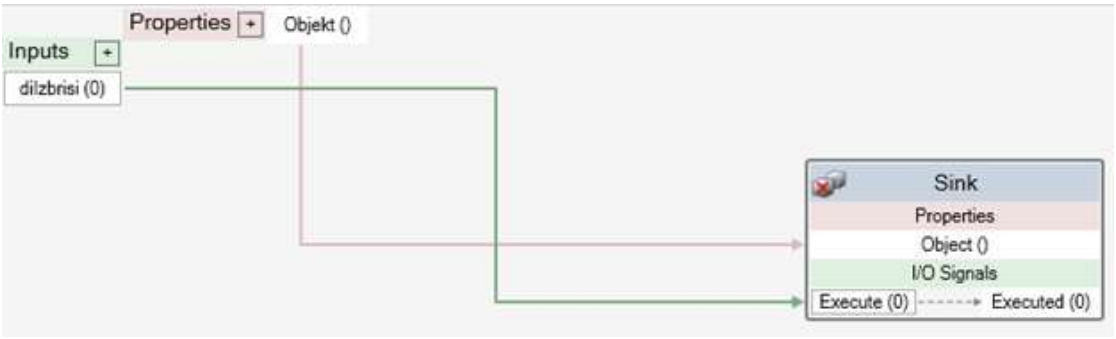


Slika 4.21. Veze unutarnjih komponenta sa ulazom, izlazima i svojstvom pametne komponente

Kada se upali ulaz „Attach“ pametne komponente „SC_Prihvatnica“, aktivira se senzor. Senzor detektira objekt s kojim je u kontaktu. Taj se objekt šalje da bude svojstvo „removeObject“ pametne komponente „SC_Prihvatnica“. Isto tako, taj se isti objekt šalje da bude svojstvo „Child“ unutarnje komponente „Attacher“. Zatim se pokrene „Attacher“, te objekt pričvrsti za robotski alat. Pomoću unutarnje komponente „LogicSRLatch“, postavi izlaz „Attached“ pametne komponente u logičku jedinicu. Istovremeno se izlaz „Dettached“ pametne komponente ugasi. Kada se ulaz „Attach“ pametne komponente ugasi, pomoću unutarnje komponente „LogicGate [NOT]“ se upali „Detacher“. On informaciju o objektu dobije preko unutarnje komponente „Attacher“. Zatim se izvrši odvajanje objekta od alata robota. Pomoću unutarnje komponente „LogicSRLatch“, „Detacher“ postavi izlaz „Dettached“ pametne komponente u logičku jedinicu. Istovremeno se izlaz „Attached“ pametne komponente ugasi. Važno je napomenuti da se u stvarnosti ne bi koristio senzor. Njegova svrha u simulaciji je dobiti podatak o kojem se objektu radi, kako bi se istim objektom moglo manipulirati.

4.7.3. Pametna komponenta za brisanje objekta

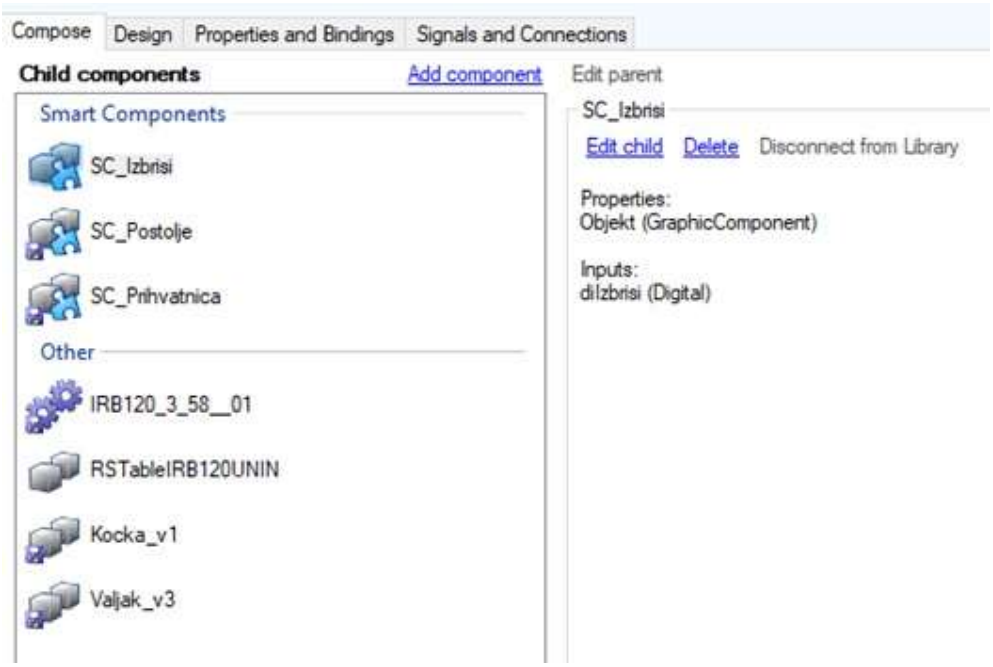
Posljednja pametna komponenta služi za brisanje stvorene kocke ili valjka. Kada kocku ili valjak hvataljka robota ostavi na krajnjoj poziciji, potrebno ih je izbrisati kako bi se napravilo mjesto za novi objekt. U stvarnosti, ovu kocku ili valjak bi pokretna traka, drugi robot ili osoba premjestili na drugo mjesto. Pametna komponenta se imenuje „SC_Izbrisi“. U ovu komponentu nije ubačen model. Ima samo jednu unutarnju komponentu, naziva „Sink“. Unutarnja komponenta „Sink“ briše definirani objekt kada je aktivirana. Pametnoj komponenti „SC_Izbrisi se dodaje ulazni signal „diIzbrisi“, te parametar „Objekt“. Ulazni signal „diIzbrisi“ se spaja sa signalom „Execute“ unutarnje komponente „Sink“. Parametar „Objekt“ pametne komponente „SC_Izbrisi“ se spaja sa parametrom „Object“ unutarnje komponente „Sink“. Unutarnji spojevi pametne komponente „SC_Izbrisi“ su prikazani na slici 4.22.



Slika 4.22.- Veze unutarnje komponente sa ulazom i izlazom pametne komponente

4.7.4. Stvaranje logike stanice

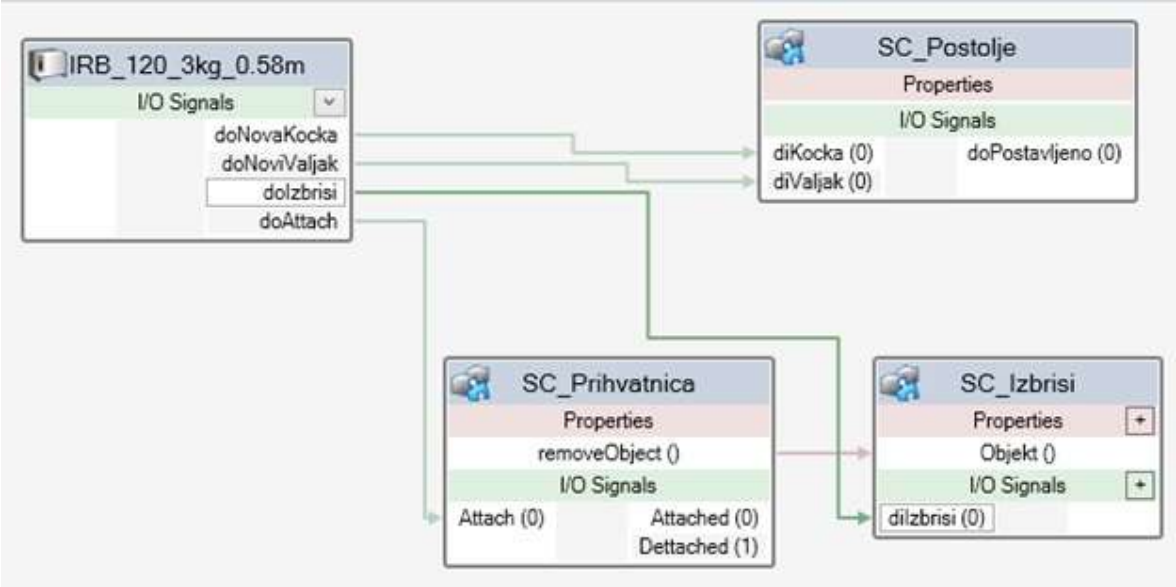
Logika stanice (eng. Station Logic) stvara se klikom na izbornik „Simulation Logic“ te odabirom značajke „Station Logic“. Izbornik „Simulation Logic“ nalazi se u „Simulation“ kartici. Značajka logike stanice radi na identičan način kao značajka pametne komponente. U logici stanice, pametne komponente imaju ulogu unutarnjih komponenata. Kontroler robota je također dodan kao unutarnja komponenta logike stanice. Na slici 4.23 su prikazane sve komponente logike stanice.



Slika 4.23.- Komponente logike stanice

Prije početka povezivanja komponenata logike stanice, potrebno je dodati signale kojima će se komponente upravljati. Kontroleru se dodaju digitalni izlazi „doNovaKocka“, „doNoviValjak“, „doIzbrisi“ i „doAttach“. Signal „doNovaKocka“ kontrolera se spaja sa signalom „diKocka“ pametne komponente „SC_Postolje“. Signal „doNoviValjak“ kontrolera se spaja sa signalom „diValjak“ pametne komponente „SC_Postolje“. Signali „doNovaKocka“ i „doNoviValjak“ upravljaju pametnom komponentom postolja, te kada se upale, stvara se nova kocka ili valjak, ovisno o tome koji je signal upaljen. Signal „doIzbrisi“ kontrolera se spaja sa signalom „diIzbrisi“ pametne komponente „SC_Izbrisi“. Kada je signal „doIzbrisi“ kontrolera upaljen, pokreće se brisanje objekta definiranog parametrom „Objekt“. Svojstvo „removeObject“ pametne komponente „SC_Prihvatnica“ se spaja sa svojstvom „Objekt“ pametne komponente „SC_Izbrisi“. Na taj način se informacija iz senzora šalje do pametne komponente „SC_Izbrisi“ te

se detektirani objekt obriše kada se pokrene pametna komponenta „SC_Izbrisi“. Signal „doAttach“ kontrolera se spaja sa signalom „Attach“ pametne komponente „SC_Prihvatnica“. Kada se upali signal „doAttach“, pokreće se spajanje objekta sa robotskim alatom pomoću pametne komponente „SC_Prihvatnica“. Način spajanja signala kontrolera sa pametnim komponentama je prikazan na slici 4.24.



Slika 4.24.- Veze izlaznih signala kontrolera sa pametnim komponentama

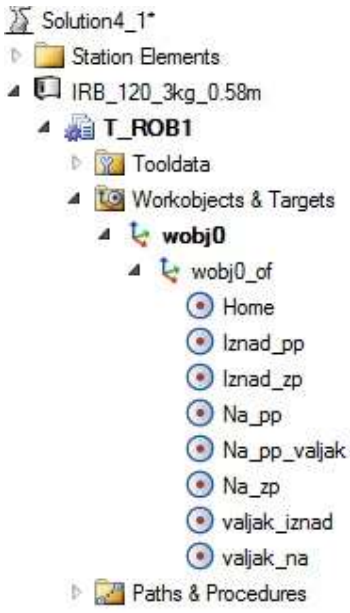
4.8. Stvaranje točaka putanja robota

Kako bi se programirale putanje robota, potrebno je definirati točke kroz koje robot prolazi. Glavna mjesta zaustavljanja robota su slijedeća:

- početna točka,
- mjesto za prihvat kocke ,
- mjesto za prihvat valjka,
- mjesto za puštanje kocke,
- mjesto za puštanje valjka.

Parametri svake točke definirani su pozicijom u prostoru (x,y,z) te orijentacijom hvataljke oko osi x,y,z. Točka se stvori odabirom naredbe „Teach Target“ koja se nalazi u kartici „Home“. Novonastala točka poprima parametre pozicije robotskog alata u trenutku stvaranja pozicije. Parametri svake točke se mogu mijenjati u „Paths&Target“ padajućem izborniku koji se nalazi u „Home“ kartici. U „Paths&Target“ izborniku se klikne na kontroler, zatim na „Workobjects & Targets“, zatim na „wobj“, pa „wobj0_of“ i konačno se odabere točka. Svakoj točki je ovdje moguće i promijeniti ime. Početna točka je pozicija iz koje se robot počinje kretati. Nazvana je

„Home“. Njeni parametri su x: 294 mm, 180 °, y: 0 mm, 0 ° z: 0 mm, 0 °. Robot kocku i valjak uzima na istom mjestu, iznad postolja za kocku i valjak. Jedina razlika je u koordinati visine, jer je valjak viši do kocke. Parametri točke prihvata kocke su x: 400 mm, 180 °, y: -317,5 mm, 0 ° z: 40 mm, 180 °. Ova točka je nazvana „Na_pp“. Parametri točke prihvata valjka su: 400 mm, 180 °, y: -317,5 mm, 0 ° z: 61.2 mm, 180 °. Ova točka je nazvana „Na_pp_valjak“. Zatim se dodaje točka iznad ove dvije točke. Parametri su x: 400 mm, 180 °, y: -317,5 mm, 0 ° z: 107,41 mm, 180 °. Naziv točke je „Iznad_pp“. Ova točka je dodana kako bi se kretanje robota do točke prihvata moglo usporiti. Zatim se dodaju točke otpuštanja kocke i valjka. Parametri točke za otpuštanje kocke su x: 395 mm, 180 °, y: 25 mm, 0 ° z: 35 mm, 180 °. Točka je nazvana „Na_zp“. Parametri točke za otpuštanje valjka su x: 300 mm, 180 °, y: 25 mm, 0 ° z: 60 mm, 180 °. Točka je nazvana „valjak_na“. Iznad ovih točaka se također dodaje po jedna točka zbog usporavanja robota. Dodaje se točka „valjak_iznad“ parametara x: 295,48 mm, 180 °, y: 24,23 mm, 0 ° z: 102,47 mm, -180 °. Zatim se dodaje točka „iznad_zp“ parametara x: 395 mm, 180 °, y: 24.23 mm, 0 ° z: 102,47 mm, -180 °. Lista svih točaka je prikazana na slici 4.25



Slika 4.25.- Lista točaka putanja robota

4.9. Programiranje robota u RAPID-u

Prije početka rada u RAPID-u potrebno je omogućiti određene postavke u postavkama kontrolera. U kartici „RAPID“ se desnim klikom na kontroler otvori izbornik gdje se odabere „Change Options…“. U kategoriji „Communication“ se upali opcija „616-1 PC Interface“, ako već nije upaljena. Zatim se u kategoriji „Engineering Tools“ upali opcija „623 – 1 Multitasking“.

Opcija „PC Interface“ omogućava spajanje kontrolera robota sa mrežom. Ovo je važno jer će robot komunicirati sa Python programom preko *socket* mrežne komunikacije. Opcija „Multitasking“ je vrlo važna jer omogućava izvođenje više RAPID zadataka (eng. tasks) istovremeno, odnosno paralelno. Dakle, ima vrlo sličnu funkciju kao Threading biblioteka koja je korištena u Python programima. Potrebna su dva RAPID zadatka. Prvi komunicira sa Python programom te na temelju toga daje instrukcije drugom programu. Drugi RAPID zadatak upravlja kretanjem robota. Zbog preglednosti sav kod nije priložen u koracima koji slijede (npr. deklaracija točaka putanja). Sav kod nalazi se u prilogu.

4.9.1. RAPID zadatak za kretanje robota

U slijedećim koracima biti će opisano stvaranje RAPID zadatka koji upravlja radom robota. Ovaj RAPID zadatak nazvan je „T_ROB1“. Unutar njega, programski kod se piše u modulu imena „Module1“. Prvi korak je sinkroniziranje postojećih točaka robota u RAPID zadatak. U „RAPID“ kartici se može pronaći „Synchronize“ naredba. Pritiskom na nju se otvori padajući izbornik na kojem se odabere „Synchronize to RAPID“. Sve ponuđeno se odabere i zatim se pritisne „OK“. Zatim se deklariraju varijable. Bitno je naglasiti da postoji razlika između „PERS“ i „VAR“ varijabli. Varijable „PERS“ su u ovom slučaju bitne jer se njihova vrijednost prenosi iz jednog RAPID zadatka u drugi. Varijabla „object“ definira koji objekt robot treba prihvatiti. Ova informacija se dobiva iz drugog RAPID zadatka koji će biti zaslužan za komunikaciju sa Python programom. Varijabla „object“ može biti samo „kocka“ ili „valjak“. Varijabla „state“ definira u kojem stanju se zadatak nalazi. Stanje može biti „detect“, u kojem robot čeka ulaznu informaciju o tome koji objekt treba prihvatiti. Stanje isto tako može biti „alarm“. To stanje traje od trenutka kad robot krene prema zadanom objektu, sve dok taj objekt ne ispusti da određenom mjestu. Varijable „object“ i „state“ vrijede za oba RAPID zadatka istovremeno pošto su „PERS“ tipa. Treća varijabla u ovom RAPID zadatku je varijabla „localState“. Ona je „VAR“ tipa što znači da vrijedi samo u ovom RAPID zadatku. Ona, kao i varijabla „object“, može biti samo „kocka“ ili „valjak“, te služi za pokretanje putanje za kocku ili putanje za valjak.

```
MODULE Module1
    PERS string object:="";
    PERS string state:="detect";
    VAR string localState:="";
```

Rutina main() je prva rutina koja se pokreće pri izvođenju zadatka. Briše podatke varijabla „object“ i „localState“, a u varijablu „state“ postavlja „detect“. Robota zatim postavlja u početnu poziciju, te signale koji su možda ostali upaljeni postavlja natrag u nulu. Završetkom ove rutine kreće se u iduću rutinu.

```
PROC main()
        object:="";
        localState:="";
        state:="detect";
        MoveL Home,v200,z100,PGN_P80_1_1\WObj:=wobj0;
        SetDO doOtvoreno,0;
        SetDO doPrihvatObjekta,0;
        Odabir_Patha;
ENDPROC
```

Rutina Odabir_Patha() čeka da se u varijablu „object“ upiše koji objekt robot treba prihvatiti. Čekanje je omogućeno naredbom „WaitUntil“. Tek kada je „state“ varijabla postavljena kao „alarm“, te je u „object“ varijabli definiran objekt, varijabla „localState“ se postavlja u „path“ te se pokreće „WHILE“ petlja. Unutar „WHILE“ petlje, programski pokazivač (eng. program pointer) skače na rutinu „Path_Kocka“ ili „Path_Valjak“, ovisno o objektu koji se nalazi u varijabli „object“.

```
PROC Odabir_Patha()
        object:="";
        WaitUntil object = "kocka" OR object = "valjak";
        IF object = "kocka" and state="alarm" THEN
            localState:="path";
            WHILE localState="path" DO
                Path_Kocka;
            ENDWHILE
        ELSEIF object = "valjak" and state="alarm" THEN
            localState:="path";
            WHILE localState="path" DO
                Path_Valjak;
            ENDWHILE
        ENDIF
        Odabir_Patha;
ENDPROC
```

Prva naredba rutine za prihvat kocke je postavljanje signala „doOtvoreno“ u jedinicu pomoću narede SetDO. Ovime se prsti alata maksimalno otvaraju. Zatim se „pulsira“ digitalni izlaz „doNovaKocka“, što stvara kocku na postolju. Naredbom „MoveL“ se robot linearno kreće do točke „Iznar_pp“. Ova naredba ima nekoliko parametara:

- robtarget – točka do koje se robot kreće,
- v (npr. v100) – označava brzinu kretanja robota,
- z (npr. z20) – označava luk koji robot radi pri kretanju, čime se skraćuje kut kretanja,
- tooldata – alat s kojim robot upravlja .

Zatim se robot sporijom brzinom pozicionira u točku „Na_pp“. Naredbom „waitTime“ se pričeka 1,5 sekunda nakon čega se digitalni izlaz „doPrihvatObjekta“ upali. Ovime se prsti alata postave u poziciju prihvata objekta. Zatim se čeka 0,3 sekunde nakon čega se upali digitalni izlaz „doAttach“ čime se objekt spaja za alat. Zatim se pričeka 1,5 sekundi te se robot počne linearno

kretati do točke „Iznad_pp“. Zatim ubrza do točke „Iznad_zp“. Do konačne točke „Na_zp“ se opet kreće sporije. Nakon 1,5 sekundi se otpušta objekt sa alata. Zatim se digitalni izlaz „doPrihvatObjekta“ stavlja u nulu. Opet se pričeka 1,5 sekundi te se robot počne kretati prema poziciji „Iznad_zp“. Nakon toga se „pulsira“ izlaz „doIzbrisi“ čime se obriše model kocke. Robot se na kraju postavi u poziciju „Iznad_pp“. Vrijednosti varijabla „object“ i „localState“ se izbrišu, a varijabla „state“ se postavi na „detect“. Ovime se robot prestao kretati te je spreman dobiti podatak o novom objektu.

```
PROC Path_Kocka()
        SetDO doOtvoreno, 1;
        PulseDO doNovaKocka;
        MoveL Iznad_pp,v100,z100,PGN_P80_1_1\WObj:=wobj0;
        MoveL Na_pp,v20,z10,PGN_P80_1_1\WObj:=wobj0;
        waitTime 1.5;
        SetDO doPrihvatObjekta, 1;
        waitTime 0.3;
        SetDO doAttach,1;
        waitTime 1.5;
        MoveL Iznad_pp,v20,z100,PGN_P80_1_1\WObj:=wobj0;
        MoveL Iznad_zp,v60,z100,PGN_P80_1_1\WObj:=wobj0;
        MoveL Na_zp,v20,z10,PGN_P80_1_1\WObj:=wobj0;
        waitTime 1.5;
        SetDO doAttach,0;
        waitTime 0.3;
        SetDO doPrihvatObjekta,0;
        waitTime 1.5;
        MoveL Iznad_zp,v50,z100,PGN_P80_1_1\WObj:=wobj0;
        PulseDO doIzbrisi;
        MoveL Iznad_pp,v50,z100,PGN_P80_1_1\WObj:=wobj0;
        object:="";
        state:="detect";
        localState:="";
ENDPROC
```

Rutina putanje „Patch_Valjak“ je identična ovoj, ali se umjesto točaka za prihvat i otpuštanje kocke upisuju točke koje odgovaraju pozicijama za valjak. Isto tako, umjesto signala vezanih isključivo za kocku, koriste se signali vezani za valjak.

4.9.2. RAPID zadatak za *socket* komunikaciju

U kartici „Controller“ se klikom na kontroler otvori padajući izbornik unutar kojega se otvori novi izbornik pod imenom „Configuration“. Stisne na „Controller“ te se odabere kartica „Task“. Desnim klikom u ovom prozoru se otvori izbornik na kojem se stisne „New Task…“. Novi zadatak se nazove „Other_task“. Unutar njega se napravi novi modul nazvan „stop_move“. Unutar RAPID-a je implementirana *socket* komunikacija. Stvaraju se varijable „socketdev serverSocket“ i „socketdev clientSocket“ za *socket* komunikaciju. Zatim se deklariraju PERS varijable „state“ i

„object“. Njihov sadržaj će se definirati u ovom zadatku. Kada se promijene ovdje, njihov sadržaj će se promijeniti i u T_ROB1 zadatku. Zatim se deklariraju varijable „tempObject“ i „alarm“.

```
MODULE stop_move
    VAR socketdev serverSocket;
    VAR socketdev clientSocket;
    PERS string state:="detect";
    PERS string object:="";
    VAR string tempObject:="";
    VAR string alarm:="";
```

U rutini „main()“ se stvaraju „serverSocker“ i „clientSocket“, te se *socket* spaja sa lokalnom IP adresom i portom koji mora biti isti kao u Python serveru. Varijabla „state“ se postavlja kao „detect“, a sadržaj varijable „object“ se briše.

```
    PROC main()
            SocketCreate serverSocket;
            SocketCreate clientSocket;
            SocketConnect serverSocket,"127.0.0.1",1025;
            state :="detect";
            object:="";
            detect;
    ENDPROC
```

U rutini „detect()“ se nalaze dvije *while* petlje. Prva se izvršava sok je zadatak u stanju detekcije, odnosno dok robot miruje, što je definirano sadržajem varijable „state“. Druga *while* petlja izvršava dok je zadatak u stanju alarma, odnosno dok se robot kreće. U prvoj *while* petlji se iz RAPID zadatka šalju *string* podatci u Python. U Pythonu se točno ovi podaci prepoznaju, te se pošalje informacija o objektu koji se nalazi na kameri. Ta se informacija sprema kao varijabla „tempObject“. Zatim se njen sadržaj kopira u PERS varijablu „object“. U Python program se zatim pošalje *string* sadržaja „Alarm“. To nema nikakvu funkciju, već je tu kako bi se u Pythonu isprintala ta poruka radi lakšeg snalaženja u izvođenju programa. Zatim se stanje zadatka postavi u „alarm“, što znači da se trenutna *while* petlja prestaje izvršavati i počinje se izvršavati druga *while* petlja. Rutina „stop_alarm“ se izvršava sve dok se ne ispuni uvjet za izlazak iz *while* petlje.

```
    PROC detect()
            WHILE state="detect" DO
                SocketSend serverSocket \Str:="Client status: Detect";
                SocketSend  serverSocket  \Str:="Client:  Waiting  for
object recognition...";
                SocketReceive serverSocket \Str:=tempObject;
                SocketSend serverSocket \Str:="Client: received";
                object:=tempObject;
                SocketSend serverSocket \Str:="Client status: Alarm";
                state :="alarm";
            ENDWHILE

            WHILE state="alarm" DO
                stop_alarm;
            ENDWHILE
```

```
            detect;
```

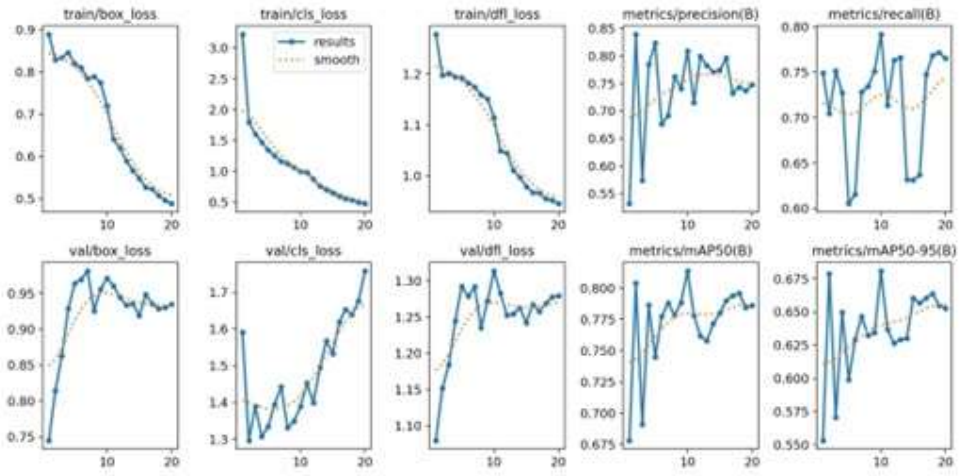
Prva naredba rutine „stop_alarm()“ šalje *string* podatak sadržaja „Check“ u Python program. Ako se osoba u tom trenutku nalazi na slici kamere, Python će u ovaj RAPID zadatak poslati podatak „alarm“. Time će se izvršiti naredba „StopMove“ koja zaustavlja kretanje robota. Zatim se opet šalje *string* sadržaja „Check“. Ako je osoba još uvijek na kameri, opet se izvršava naredba „StopMove“. Ovo ne utječe na robota pošto je već zaustavljen. Ovaj proces se ponavlja sve dok se osoba ne makne sa kamere. Tek onda Python šalje *string* sadržaja „none“, ali tek nakon što primi *string* sadržaja „Check“. Ovime se ne ispunjava *IF* izjava koja dovodi do zaustavljanja robota. Izvršava se naredba „StartMove“ te se robot nastavlja kretati.

```
    ENDPROC
    PROC stop_alarm()
            !SocketReceive serverSocket \Str:=alarm;
            !IF alarm = "alarm" THEN
            !    StopMove \Quick;
            !ELSE
            !    StartMove;
            !ENDIF
             SocketSend serverSocket \Str:="Check";
             SocketReceive serverSocket \Str:=alarm;
             IF alarm = "alarm" THEN
                StopMove \Quick;
             ELSE
                StartMove;
             ENDIF
    ENDPROC
ENDMODULE
```

5. Analiza rezultata

5.1. Analiza YOLO treniranja

Istrenirani YOLOv8 model neuronske mreže radi dovoljno dobro za potrebe ovog programa, no na temelju dobivenih rezultata (slika 5.1) se vidi da postoji mjesta za poboljšanje.



Slika 5.1.- Rezultati treniranja YOLO modela

Svi parametri vezani za treniranje padaju, te se po izgledu krivulja vidi da bi njihov pad nastavio i sa daljnjim treniranjem. Sa druge strane, parametri vezani uz potvrđivanje ili rastu ili su naglo porasli na početku te se zasitili. Ovo nije dobra pojava. Optimalno bi bilo da grafovi ovih parametara pokazuju pad kao i kod parametara treniranja. Ovo znači da je modelova sposobnost detektiranja na sadržaju kojeg nije vidio sve gora, dok je za sadržaj sa kojim je upoznat sve bolja. Razlog ovoga je pojava zvana kao „Overfitting“ . Korištenjem raznolikijeg sadržaja treniranja bi moglo riješiti ovaj problem. To uključuje slike objekata iz drugih kutova, sa različitih udaljenosti, na različitim osvjetljenjima, sa drugačijim pozadinama pa čak i u drugačijim bojama (na primjer crno bijelo). Ostali parametri ili rastu ili su relativno visoki što je dobro.

5.2. Analiza rada robota

Robota je potrebno zaustaviti čim se osoba pojavi na kameri. Dakle, postoji poveznica između rada robota i brzine detekcije objekata na kameri. Vremenska razlika između pojavljivanja osobe na kameri i zaustavljanja robota je primjetna. To znači da se robot ne zaustavi u istom trenutku kada se pojavi osoba na kameri, već postoji mali vremenski razmak. Taj vremenski razmak je većim dijelom izazvan brzinom osvježavanja slika kamere. Dakle, što više slika u sekundi kamera

ima, to se brže pošalje podatak o osobi na kameri. Isprobavanjem kamere izvan programa može se zaključiti da problem ne leži u kameri već u optimizaciji kamere u programu.

6. Zaključak

Treniranje modela neuronske mreže za potrebe ovoga rada nije teško, ali je trebalo dosta vremena za slikanje i označavanje svake slike posebno. Za slučajeve kao u ovom radu, gdje su neki objekti specifični i nije moguće naći skupove slika sa oznakama na internetu, proces treniranja može uzeti puno vremena jer je svaku sliku potrebno ručno klasificirati i spremiti.

U ovom radu je objašnjen način interpretacije rezultata dobivenih nakon treninga modela neuronske mreže. Znanjem interpretiranja rezultata treninga može se zaključiti je li model mreže spreman za korištenje, a ako nije, kako ga popraviti i bolje istrenirati. Iz rezultata se vidjelo da model nije savršen, te da postoji još mjesta za optimizaciju u području treninga.

Brzina detekcije je zadovoljavajuća, no činjenica je da nije optimalna. Količina slika u sekundi kamere najviše utječe na brzinu detekcije objekata. Ukoliko bi se broj FPS-a kamere mogao optimizirati u programu, brzina detekcije bi se znatno povećala, a time i vrijeme reagiranje robota na promjene u kadru kamere.

Socket komunikacija je izvrsna za ovakav zadatak, pošto je ugrađena u RAPID programski jezik i jednostavna je za programiranje. U ovom primjeru, Python i RAPID programi nalaze se na istom uređaju, te brzina i postojanje internetske veze ne utječu na *socket* komunikaciju. U stvarnosti, ako se radi o dva različita uređaja na različitim mrežama, internetska veza može znatno utjecati komunikaciju između programa.

Simulacija u RobotStudio-u pokazuje da bi ovakav program, u teoriji, dobro radio u stvarnosti. No u stvarnosti postoji mogućnost javljanja drugih problema poput točnosti definiranih koordinata postolja, kvalitete prihvata objekata i slično. Isto tako, postoji mogućnost da postavka „Multitasking“ ne funkcionira, a ako funkcionira, postoji mogućnost da rad sa postavkom „Multitasking“ ne bude optimalan, što bi izazvalo još veća vremenska kašnjenja. Na primjer, možda programski pokazivači dvaju zadataka RAPID-a, koji se izvode istovremeno, ne budu sinkronizirani kao u simulaciji. Isto tako, *socket* komunikacija u stvarnosti možda nije brza kao u simulaciji. Također, brzina rada kontrolera u stvarnosti možda nije ista kao u simulaciji što bi dovelo do većih vremenskih zastajanja.

7. Literatura

[1] https://www.researchgate.net/figure/Industrial-revolutions-in-human-history-The-first-industrial-revolution-used-water-and_fig5_348351406, dostupno 02.09.2024

[2] https://www.mathworks.com/discovery/deep-learning.html#whyitmatters, dostupno 02.09.2024.

[3] https://www.v7labs.com/blog/yolo-object-detection#what-is-yolo, dostupno 02.09.2024.

[4] https://docs.ultralytics.com/, dostupno 02.09.2024.

[5] https://medium.com/analytics-vidhya/image-classification-vs-object-detection-vs-image-segmentation-f36db85fe81, dostupno 02.09.2024

[6] https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/, dostupno 02.09.2024

[7] https://towardsdatascience.com/what-is-average-precision-in-object-detection-localization-algorithms-and-how-to-calculate-it-3f330efe697b, dostupno 03.09.2024.

[8] https://docs.ultralytics.com/guides/yolo-performance-metrics/#introduction, dostupno 03.09.2024.

[9] https://docs.python.org/3/tutorial/index.html, dostupno 02.09.2024

[10] https://www.python.org/doc/essays/blurb/?external_link=true, dostupno 02.09.2024

[11] https://ai.meta.com/tools/pytorch/, dostupno 02.09.2024

[12] https://pytorch.org/docs/stable/index.html, dostupno 02.09.2024

[13] https://www.ibm.com/docs/en/i/7.2?topic=communications-socket-programming, dostupno 04.09.2024.

[14] https://www.ibm.com/docs/en/i/7.2?topic=programming-how-sockets-work, dostupno 04.09.2024.

[15] https://www.geeksforgeeks.org/types-of-socket/, dostupno 04.09.2024.

[16] https://new.abb.com/products/robotics/robotstudio, dostupno 04.09.2024.

[17] https://moodle.srce.hr/2023-2024/mod/resource/view.php?id=3583727, dostupno 28.8.2024

Popis slika

Prilozi:

Ispis programa train.py

Ispis programa prediction.py

Ispis programa interface.py

Ispis programa server.py

Ispis RAPID modula RAPID T_ROB1/Module1

Ispis RAPID modula RAPID Other_Task/stop_move

Program train.py

```
from ultralytics import YOLO

model = YOLO('yolov8m.pt')

if __name__ == '__main__':
                                                results                    =
model.train(data='c://Users//pudin//runs//detect//Detection//data.yml
', epochs=20, workers=1)
```

Program prediction.py

```
from ultralytics import YOLO
from server import update
import cv2

cam = cv2.VideoCapture(1)
#model = YOLO(r'c:\Users\pudin\runs\detect\train2\weights\last.pt')
model = YOLO(r'train2\weights\last.pt')
obj_list = []

#Sve sta se ovdje dogada je citanje objekata u frameovima i pozivanje
update funkcije da se lista objekata osvjezi u server.py
#Svo upravljanje detektiranim objektima se dogada u server.py
def predict():
    global model
    global cam
    global obj_list

    while True:
        _,frame = cam.read()
          results = model.predict(source=frame, show=True, iou=0.5,
conf=0.3)#[0]
        obj_frame = []
        for r in results:                                #-> Prolazi kroz
svaki  pojedini rezultat
            for b in r.boxes:                            #-> Prolazi kroz
svaki box koji se nalazi na kameri
                obj_index = int(b.cls)                    #-> Definiranje
indexa objekta ciji se box nalazi na kameri
                obj_name = model.names[int(b.cls)]   #-> Definiranje
imena objekta pomocu indexa
                print(obj_index,obj_name)                 #-> Printa broj
klase koja se nalazi u boxu
               obj_frame.append(obj_name)

        obj_list = obj_frame
        update(obj_list)                               #-> Poziva funkciju
iz server.py sa listom objekata u kadru
```

Program interface.py

```
from PyQt6.QtWidgets import QApplication, QWidget, QPushButton,
QVBoxLayout
import sys
import server
import prediction
import threading

class MyApp(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('Window')
        self.resize(900,700)
        layout = QVBoxLayout()
        self.setLayout(layout)

            server_start = QPushButton("Pokreni server",clicked =
thread3.start)
        kamera = QPushButton("Kamera", clicked = thread2.start)

        layout.addWidget(server_start)
        layout.addWidget(kamera)

thread2 = threading.Thread(target=prediction.predict)
thread3 = threading.Thread(target=server.receive)


if __name__ == "__main__": # ne izvrsi se cijeli importani modul odma
    app = QApplication(sys.argv)
    app.setStyleSheet('''
        QWidget {
                    font-size: 25px;
        }
        QPushButton {
                    font-size: 20px;
        }
                    ''')
    window = MyApp()
    window.show()
    app.exec()
```

Program server.py

```
import socket
import threading
import time
import sys

host = '127.0.0.1'
port = 1025

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind((host,port))
server.listen()

clients = []
received_data = ''
updated_objects = []

def update(objects):
    #Updatea objekte u kadru, poziva se iz prediction.py programa gdje
se stalno detektiraju objekti. Svo upravljanje objekata pocinje odavde.
    global updated_objects
    updated_objects= objects

def receive():
    #Ova funkcija se pokrene u Threadu u prediction.py
    #Zatim i ova funkcija pokrece thread -> handle funkcija, i thread
define_obj
    print('Server is running...')
    print("recieving...")
    global clients
    while len(clients) == 0:
        client, adress = server.accept()
        print(f'Connected with {str(adress)}')
        clients.append(client)
        thread1 = threading.Thread(target=handle, args=(client,))
        thread4 = threading.Thread(target=define_obj)
        thread1.start()
        thread4.start()
        sys.exit()

def handle(client):
    #Cijelo vrijeme primanje podataka iz RAPIDA.
    #Slanje podataka na temelju primljene informacije.
    #Ovdje program salje povratnu informaciju ako osoba dode u frame.
    global updated_objects
    while True:
        global received_data
        received_data = client.recv(1024).decode('ascii')
        if received_data == "Check":
            if "person" in updated_objects:
                clients[0].send("alarm".encode('ascii'))
                print("sent...")
            else:
                clients[0].send("none".encode('ascii'))
                print("sent...")
        else:
            print(received_data)
```

```
def define_obj():
    #Ova funkcija sluzi za automatsko detektiranje objekta.
    #Posto je u threadu, cijelo vrijeme bez delaya detektira u kojem
je procesu program na temelju povratne informacije received_data iz
RAPIDa
    #U zasebnom je Threadu jer se inace nakon nekog vremena stvori
ogroman delay izmedu programa u Pythonu i RAPIDu
    #NPR funkcija alarma kasni 10+ sekundi
    global updated_objects
    while True:
        global received_data #Nalazi se ovdje jer se inace nebi uopce
refreshala posto nebi bila u While petlji
        print(f'received data is {received_data}')
        if   received_data   ==   "Client:   Waiting   for   object
recognition..." or received_data == "Client: received" :
            sent = False
            while sent == False:
                if  ("kocka"  in  updated_objects  or  "valjak"  in
updated_objects) and len(updated_objects)==1: #U kadru smije biti samo
kocka ili samo valjak.
                    print(f'{updated_objects[0]} has been found in the
frame!')
                    time.sleep(4) #Ovaj thread spava na 4 sekunde
                    if  ("kocka"  in  updated_objects  or  "valjak"  in
updated_objects) and len(updated_objects)==1: #U kadru smije biti samo
kocka ili samo valjak.
                        sent = True
                        det_obj()
                    else:
                        print("Object has been moved.")
                else:
                    print("No objects detected...")

def det_obj():
    #Slanje informacije robotu, dal se radi o kocki ili valjku kad on
treba prihvatit objekt. Nista drugo.
    #Ova se funkcija poziva is define_obj funkcije
    global updated_objects
    print(updated_objects)
    if len(updated_objects)>1:
        print("Multiple objects detected!")
    elif len(updated_objects) == 0:
        print("No objects in frame.")
    elif updated_objects[0] == "kocka":
        clients[0].send("kocka".encode('ascii'))
    elif updated_objects[0] == "valjak":
        clients[0].send("valjak".encode('ascii'))
```

```
MODULE Module1
    PERS string object:="";
    PERS string state:="detect";
    VAR string localState:="";
    CONST                                                    robtarget
Home:=[[294,0,475],[0,0,1,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09
,9E+09]];
    CONST                  robtarget                  Iznad_pp:=[[400,-
317.5,107.409652734],[0.000000049,-0.000000017,1,0.000000024],[-1,0,-
1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST                                                    robtarget
Iznad_zp:=[[394.996327091,24.230323586,102.472604494],[0.000000036,-
0.000000015,1,0.000000022],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9
E+09]];
     CONST     robtarget     Na_pp:=[[400,-317.5,40],[0,0,1,0],[-1,0,-
1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
     CONST                                                   robtarget
Na_zp:=[[394.996,25,35],[0,0,1,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,
9E+09,9E+09]];
    CONST                                                    robtarget
valjak_iznad:=[[295.479297425,24.230319007,102.472554196],[0.00000003
6,-
0.000000011,1,0.000000022],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9
E+09]];
    CONST                                                    robtarget
valjak_na:=[[300,25,60],[0,0,1,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,
9E+09,9E+09]];
    CONST   robtarget   Na_pp_valjak:=[[400,-317.5,61.2],[0,0,1,0],[-
1,0,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];


    PROC main()
            object:="";
            localState:="";
            state:="detect";
            MoveL Home,v200,z100,PGN_P80_1_1\WObj:=wobj0;
            SetDO doOtvoreno,0;
            SetDO doPrihvatObjekta,0;
            Odabir_Patha;
    ENDPROC

    PROC Odabir_Patha()
            object:="";
            WaitUntil object = "kocka" OR object = "valjak";
            IF object = "kocka" and state="alarm" THEN
                localState:="path";
                WHILE localState="path" DO
                    Path_Kocka;
                ENDWHILE
            ELSEIF object = "valjak" and state="alarm" THEN
                localState:="path";
                WHILE localState="path" DO
                    Path_Valjak;
                ENDWHILE
            ENDIF
            Odabir_Patha;
```

```
    ENDPROC

    PROC Path_Kocka()
            SetDO doOtvoreno, 1;
            PulseDO doNovaKocka; !Na kameri je kocka -> Stvori kocku
            MoveL Iznad_pp,v100,z100,PGN_P80_1_1\WObj:=wobj0;
            MoveL Na_pp,v20,z10,PGN_P80_1_1\WObj:=wobj0;
            waitTime 1.5;
            SetDO doPrihvatObjekta, 1; !Pomak mehanizma -> Prsti alata
se pomaknu u poziciju prihvata
            waitTime 0.3;                !Treba biti vremenski delay
izmedu jer se inace objekt pomakne
            SetDO doAttach,1;          !Upali se "attach" -> Objekt se
"zalijepi" za alat
            waitTime 1.5;
            MoveL Iznad_pp,v20,z100,PGN_P80_1_1\WObj:=wobj0;
            MoveL Iznad_zp,v60,z100,PGN_P80_1_1\WObj:=wobj0;
            MoveL Na_zp,v20,z10,PGN_P80_1_1\WObj:=wobj0;
            waitTime 1.5;
            SetDO doAttach,0;
            waitTime 0.3;
            SetDO doPrihvatObjekta,0;
            waitTime 1.5;
            MoveL Iznad_zp,v50,z100,PGN_P80_1_1\WObj:=wobj0;
            PulseDO doIzbrisi;
            !SetDO doOtvoreno, 0; -> Ne treba uopce stavljat u nulu,
kontrolira se sa doPrihvatKocke/doPrihvatValjka
            MoveL   Iznad_pp,v50,z100,PGN_P80_1_1\WObj:=wobj0;   !->
Bolje je ne imati ovo ovdje jer se stvori pre veliki delay izmedu
pointera programa i robota
            object:="";
            state:="detect";
            localState:="";
    ENDPROC

    PROC Path_Valjak()
            SetDO doOtvoreno, 1;
            PulseDO doNoviValjak; !Na kameri je valjak -> Stvori valjak
            MoveL Iznad_pp,v100,z100,PGN_P80_1_1\WObj:=wobj0;
            MoveL Na_pp_valjak,v20,z10,PGN_P80_1_1\WObj:=wobj0;
            waitTime 1.5;
            SetDO doPrihvatObjekta, 1;
            waitTime 0.3;
            SetDO doAttach,1;
            waitTime 1.5;
            MoveL Iznad_pp,v20,z100,PGN_P80_1_1\WObj:=wobj0;
            MoveL valjak_iznad,v60,z100,PGN_P80_1_1\WObj:=wobj0;
            MoveL valjak_na,v20,z10,PGN_P80_1_1\WObj:=wobj0;
            waitTime 1.5;
            SetDO doAttach,0;
            waitTime 0.3;
            SetDO doPrihvatObjekta,0;
            waitTime 1.5;
            MoveL valjak_iznad,v50,z100,PGN_P80_1_1\WObj:=wobj0;
            PulseDO doIzbrisi;
            !SetDO doOtvoreno, 0; -> Ne treba uopce stavljat u nulu,
kontrolira se sa doPrihvatKocke/doPrihvatValjka
            MoveL Iznad_pp,v50,z100,PGN_P80_1_1\WObj:=wobj0;
```

```
            object:="";
            state:="detect";
            localState:="";
    ENDPROC
ENDMODULE
```

RAPID Other_Task/stop_move

```
MODULE stop_move
    VAR socketdev serverSocket;
    VAR socketdev clientSocket;
    PERS string state:="detect";
    PERS string object:="";
    VAR string tempObject:="";
    VAR string alarm:="";
    !VAR string localState:="";

    PROC main()
            SocketCreate serverSocket;
            SocketCreate clientSocket;
            SocketConnect serverSocket,"127.0.0.1",1025;
            state :="detect";
            object:="";
            detect;
    ENDPROC
    PROC detect()
            WHILE state="detect" DO
                SocketSend serverSocket \Str:="Client status: Detect";
                SocketSend  serverSocket  \Str:="Client:  Waiting  for
object recognition...";
                SocketReceive serverSocket \Str:=tempObject;
                SocketSend serverSocket \Str:="Client: received";
                object:=tempObject;
                SocketSend serverSocket \Str:="Client status: Alarm";
                state :="alarm";
            !SocketSend serverSocket \Str:="ALARM!";
            !SocketSend serverSocket \Str:="Trying again...";
            ENDWHILE

            WHILE state="alarm" DO
                stop_alarm;
            ENDWHILE
            detect;

    ENDPROC
    PROC stop_alarm()
            !SocketReceive serverSocket \Str:=alarm;
            !IF alarm = "alarm" THEN
            !    StopMove \Quick;
            !ELSE
            !    StartMove;
            !ENDIF
             SocketSend serverSocket \Str:="Check";
             SocketReceive serverSocket \Str:=alarm;
             IF alarm = "alarm" THEN
                StopMove \Quick;
             ELSE
                StartMove;
             ENDIF
    ENDPROC
ENDMODULE
```

Sveučilište
Sjever

SVEUČILIŠTE
SJEVER

IZJAVA O AUTORSTVU

Završni/diplomski/specijalistički rad isključivo je autorsko djelo studenta koji je isti izradio te student odgovara za istinitost, izvornost i ispravnost teksta rada. U radu se ne smiju koristiti dijelovi tuđih radova (knjiga, članaka, doktorskih disertacija, magistarskih radova, izvora s interneta, i drugih izvora) bez navođenja izvora i autora navedenih radova. Svi dijelovi tuđih radova moraju biti pravilno navedeni i citirani. Dijelovi tuđih radova koji nisu pravilno citirani, smatraju se plagijatom, odnosno nezakonitim prisvajanjem tuđeg znanstvenog ili stručnoga rada. Sukladno navedenom studenti su dužni potpisati izjavu o autorstvu rada.

Ja, Marko Hmelik *(ime i prezime)* pod punom moralnom, materijalnom i kaznenom odgovornošću, izjavljujem da sam isključivi autor/~~ica~~ završnog/~~diplomskog/specijalističkog~~ *(obrisati nepotrebno)* rada pod naslovom Rad robota na temelju prepoznavanja dijelova snimljenih kamerom *(upisati naslov)* te da u navedenom radu nisu na nedozvoljeni način (bez pravilnog citiranja) korišteni dijelovi tuđih radova.

Student/ica:
(upisati ime i prezime)

MH
(vlastoručni potpis)

Sukladno članku 58., 59. i 61. Zakona o visokom obrazovanju i znanstvenoj djelatnosti završne/diplomske/specijalističke radove sveučilišta su dužna objaviti u roku od 30 dana od dana obrane na nacionalnom repozitoriju odnosno repozitoriju visokog učilišta.

Sukladno članku 111. Zakona o autorskom pravu i srodnim pravima student se ne može protiviti da se njegov završni rad stvoren na bilo kojem studiju na visokom učilištu učini dostupnim javnosti na odgovarajućoj javnoj mrežnoj bazi sveučilišne knjižnice, knjižnice sastavnice sveučilišta, knjižnice veleučilišta ili visoke škole i/ili na javnoj mrežnoj bazi završnih radova Nacionalne i sveučilišne knjižnice, sukladno zakonu kojim se uređuje umjetnička djelatnost i visoko obrazovanje.