# Validacija algoritma za brzu detekciju i praćenje objekta

**Dobrić, Bruno**




SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

# DIPLOMSKI RAD

**Bruno Dobrić**

Zagreb, 2021.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

# DIPLOMSKI RAD

Mentor:

Doc. dr. sc. Marko Švaco, dipl. ing.

Student:

Bruno Dobrić

Zagreb, 2021.

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

Zahvaljujem svom mentoru, doc. dr. sc. Marku Švaci na pruženoj prilici za izradu diplomskog rada u Centru izvrsnosti za robotske tehnologije te na usmjeravanju i stručnim savjetima.

Konačno, zahvaljujem svojoj obitelji na moralnoj i financijskoj potpori tijekom studiranja te prijateljima i kolegama na motivaciji i podršci.

Bruno Dobrić

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

| Sveučilište u Zagrebu<br>Fakultet strojarstva i brodogradnje | |
|---|---|
| Datum: | Prilog: |
| Klasa: 602-04/21-6/1 | |
| Ur. broj: 15-1703-21 | |

# DIPLOMSKI ZADATAK

Student: **BRUNO DOBRIĆ** Mat. br.: 0035201866

Naslov rada na hrvatskom jeziku: **Validacija algoritma za brzu detekciju i praćenje objekata**

Naslov rada na engleskom jeziku: **Validation of an algorithm for fast object detection and tracking**

Opis zadatka:

Algoritmi za brzi pronalazak (eng. detection) i praćenje (eng. tracking) objekata i ljudi u dvodimenzionalnim slikama dosegnuli su vrlo visoki stupanj razvoja. Korištenjem računalnih modela dobivenih strojnim učenjem moguće je klasificirati naučeni objekt ili osobu u 2D slici unutar vremenskog intervala od 10 ms.
U sklopu ovog rada potrebno je izgraditi neuronsku mrežu koja će implementirati tzv. YOLO (eng. You Only Look Once) model za detekciju i praćenje objekata od interesa, kojeg karakterizira vrlo visoka brzina prepoznavanja naučenih objekata. Neuronsku mrežu treba naučiti (trenirati) na slikama medicinskog robota dostupnog u laboratoriju. Ovaj korak podrazumijeva pripremu velikog broja slika za učenje, kako bi YOLO model robusno prepoznavao medicinskog robota iz različitih kutova i iz različitih udaljenosti. Za prikupljanje 2D slika potrebno je koristiti streovizijski sustav dostupan u laboratoriju.
Detekciju i praćenje medicinskog robota na slikama potrebno je implementirati u ROS okruženju (eng. Robot Operating System). Neuronska mreža treba u realnom vremenu pružati informaciju je li objekt detektiran te koja je pozicija težišta objekta na slici. Također treba pružiti procjenu trodimenzionalne lokacije objekta u koordinatnom sustavu stereovizijske kamere.
Rad neuronske mreže potrebno je validirati u laboratoriju, u scenariju traženja i lokalizacije medicinskog robota.
U radu je potrebno navesti korištenu literaturu i eventualno dobivenu pomoć.

Zadatak zadan: 6. svibnja 2021.

Rok predaje rada: 8. srpnja 2021.

Predviđeni datum obrane: 12. srpnja do 16. srpnja 2021.

Zadatak zadao: doc. dr. sc. Marko Švaco

Predsjednica Povjerenstva: prof. dr. sc. Biserka Runje

# SADRŽAJ

# POPIS SLIKA

# POPIS TABLICA

# SAŽETAK

Tema ovog diplomskog rada je validacija algoritma za brzu detekciju i praćenje objekta. Objekt o kojem se radi je neuronavigacijski sustav RONNA razvijen u Centru izvrsnosti za robotske tehnologije pri Fakultetu strojarstva i brodogradnje Sveučilišta u Zagrebu. RONNA se koristi za neurokirurške operacije, a transport od spremišta do operacijske sale zasad se obavlja ručno. Cilj ovog zadatka je razvoj algoritma za detekciju i praćenje RONNA-e koji bi se vrtio na mobilnom robotu koji bi autonomno prevozio RONNA-u s mjesta gdje se nalazi do mjesta gdje je potrebna. Na taj način uštedjelo bi se vrijeme liječnika i medicinskog osoblja bolnice koje ne bi morali potrošiti na taj prijevoz. U radu je dan pregled područja algoritama za detekciju i praćenje objekata, opisana je korištena oprema te je detaljno opisan proces treniranja YOLO v5 neuronske mreže vlastitim podatcima.

Ključne riječi: detekcija objekta, praćenje objekta, neuronske mreže, YOLO v5

## SUMMARY

The topic of this Master`s thesis is validation of an algorithm for fast object detection and tracking. The object is RONNA robotic system which is developed in Regional Center of Excellence for Robotic Technology at the Faculty of mechanical Engineering and Naval Architecture, University of Zagreb. RONNA is used in neurosurgery and transport from the storehouse to the operating room still takes place manually. The aim of this thesis is to develop algorithm for detection and tracking RONNA robotic system for an autonomous mobile robot that would transport RONNA. This would save time for doctors and medical staff at the hospital who would not have to spend on this transport. The paper presents an overview of the field of algorithms for detection and tracking of objects, describes the equipment used and describes in detail the process of training the YOLO v5 neural network with its own data.

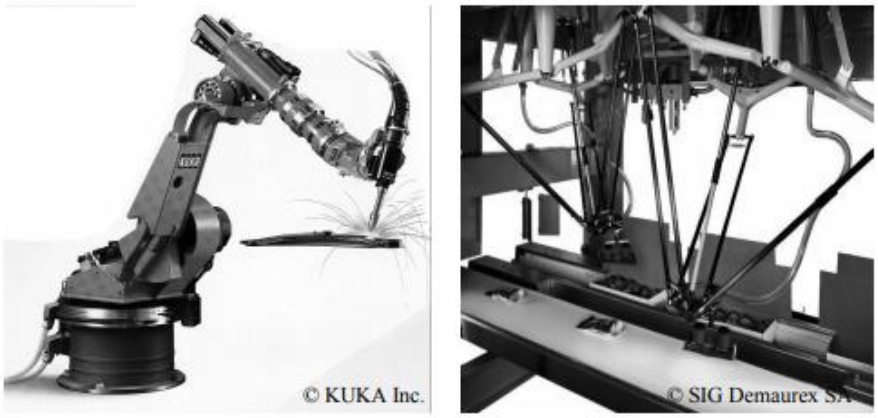
Key words: object detection, object tracking, neural networks, YOLO v5

# 1. UVOD

Još od davnina ljudi su željeli napraviti stroj koji je sposoban i inteligentan. Ta želja u današnje vrijeme je postala stvarnost. Koncept današnjih robota nastao je na temelju brojnih kreativnih povijesnih djela. Vodeni sat Clepsydra kojeg su Babilonci izradili 1400. godine prije Krista jedan je od prvih automatiziranih mehaničkih sustava. Dobar temelj današnjih sustava i robota bili su i automatizirano kazalište Herona iz Aleksandrije, Al-Jazarijevi humanoidni strojevi pokretani vodom i razne ingeniozne konstrukcije Leonarda da Vincija. Pojam robot dolazi od slavenske riječi robota što označava subordinirani posao, a prvi put ga je upotrijebio češki pisac Karel Čapek 1920. godine. [1]

Pojam robota u današnje vrijeme jako je širok i proteže se od jednostavnih uređaja namijenjenih za pomoć čovjeku u obavljanju jednostavnih radnji pa sve do čovjekolikih robota čija je svrha prezentacija znanstvenih dostignuća u području umjetne inteligencije. Robotima se tako smatraju razni kuhinjski uređaji, mobilni usisivači i slično, ali i industrijski roboti koji se koriste u proizvodnim pogonima gdje zamjenjuju ljude pri obavljanju repetitivnih i opasnih poslova. Stalni razvoj robotike kao znanosti posljednjih godina kontinuirano podiže kvalitetu ljudskog života pa u budućnosti tako više neće biti potrebe za ljudskom radnom snagom na nisko plaćenim poslovima nego će se svi ljudi baviti kreativnim poslovima.

Robotika je doživjela najveći uspjeh u svijetu industrijske proizvodnje. Robotska ruka pričvršćena na određenu poziciju na proizvodnoj traci može se gibati velikom brzinom i s visokom točnošću što je idealno za ponavljajuće poslove kao točkasto zavarivanje, bojanje i pakiranje proizvoda. [2]




**Slika 1. Točkasto zavarivanje robotom KUKA i pakiranje čokolada paralelnim robotom Delta SIG Demaurex SA [2]**

## 1.1. Općenito o mobilnim robotima

Mobilna robotika je relativno mlado područje. Svoje začetke duguje brojnim inženjerskim i znanstvenim disciplinama, od strojarstva, elektrotehnike i elektronike pa do računalnih, kognitivnih i društvenih znanosti. [2] Pod pojmom mobilnog robota podrazumijevaju se uređaji najčešće na kotačima koji su sposobni samostalno se kretati od početne do krajnje pozicije pritom izbjegavajući prepreke u prostoru. Kako bi to postigli opremljeni su raznim senzorima poput stereovizijskih sustava, kamera, lidara, radara i drugih. Takvi roboti najčešće služe za prenošenje nekih stvari s jednog mjesta na drugo umjesto čovjeka. Najčešća primjena mobilnih robota je u skladištima gdje se koriste za transport paleta koje bi u suprotnom prevozio čovjek, ručno ili na viličaru. Jedan takav potpuno autonomni mobilni robot razvila je hrvatska tvrtka Gideon Brothers, a primjenjuje se u skladištima gdje prevozi palete. Robot tvrtke Gideon Brothers prikazan je na slici 2.

**Slika 2. Potpuno autonomni mobilni robot tvrtke Gideon Brothers [3]**

Mobilni robot treba lokomotorni mehanizam koji će mu omogućiti kretanje kroz okruženje bez granica. Postoji mnogo različitih načina na koji se roboti mogu gibati, a izbor odgovarajućeg načina gibanja važan je aspekt u konstrukciji mobilnog robota. U laboratorijima se provode istraživanja robota koji mogu hodati poput ljudi, skakati, trčati, plivati, letjeti i naravno, kotrljati se na kotačima. Najveći dio mehanizama za kretanje robota inspiriran je njihovim biološkim kolegama. Ipak, postoji iznimka kada se radi o robotima na kotačima. To je u potpunosti izum čovjeka koji na ravnim površinama postiže izrazito dobre rezultate. Biološki sustavi, s druge strane, uspješno savladavaju bilo kakve terene, ali njih je izrazito teško replicirati iz više razloga. Prvi razlog je taj što se kod bioloških sustava lagano postiže kompleksna struktura dijeljenjem stanica, a kod izradbenih sustava svaki dio bi se morao proizvoditi posebno i to bi jako povisilo troškove proizvodnje. Kao drugo, biološka stanica je mikroskopske veličine što

omogućuje veliku minimizaciju. S tako malim dimenzijama i malom masom, biološki sustavi mogu postići nivo robusnosti koji je nedostižan za umjetne sustave. Konačno, biološki sustavi pohrane energije i mišićni sustavi koje koriste životinje i insekti omogućuju moment, brzinu reakcije i efikasnost pretvorbe koja je još uvijek nedostižna za umjetne sustave. [2]

## 1.2. Motivacija za izradu zadatka

RONNA G4 je robotski neuronavigacijski sustav temeljen na robotskoj ruci koji služi za minimalno invazivne stereotaktičke procedura kao što su biopsije, stereoelektroencefalografije, operacije epilepsije, duboke stimulacije mozga i resekcije tumora. RONNA se može koristiti kao sustav s jednom robotskom rukom ili s dvije robotske ruke. U prvom slučaju se radi o stereotaktičkoj neuronavigaciji koja služi kao pomoć pri navigaciji neurokirurgu, dok u konfiguraciji s dvije robotske ruke ima sposobnost autonomnih invazivnih operacijskih zadataka kao bušenje lubanje, umetanje sonde ili igle i slično. RONNA-u karakterizira potpuno automatska procedura registracije pacijenta , planiranje pozicioniranja robota, precizno vođenje instrumenta i autonomno bušenje lubanje. Upotreba RONNA-e za stereotaktičke procedure kod operacija na mozgu doprinijela je skraćivanju vremena operacije, smanjenju invazivnosti procedure, bržem oporavku pacijenta i boljoj iskoristivosti bolničkih resursa. [4]


**Slika 3. Robotski neuronavigacijski sustav RONNA G4 [5]**

RONNA sustav razvijen je u Regionalnom centru izvrsnosti za robotske tehnologije pri Fakultetu strojarstva i brodogradnje, a koristi se u Kliničkoj bolnici Dubrava u Zagrebu. Budući da se RONNA koristi u operacijskoj sali u kojoj se izvode i druge operacije, za vrijeme dok se ne koristi mora biti parkirana u skladištu izvan sale. Iako za pomicanje RONNA ima kotače, nije ju lagano svaki put ručno dovoziti u salu, a osim toga takvo dovoženje oduzima vrijeme,

posebno ako skladište gdje je parkirana nije blizu operacijske sale. Kako bi se riješio taj problem, ideja je razvoj autonomnog mobilnog robota koji bi vizijskim sustavom locirao RONNA-u u prostoriji, otišao po nju i dovezao ju na određeno mjesto. Jedan dio razvoja takvog mobilnog robota podrazumijeva izradu algoritma za brzu detekciju i praćenje RONNA-e.


**Slika 4. Automatska lokalizacija pacijenta stereovizijskim sustavom RONNAstereo [4]**

## 1.3. Metodologija rada

U okviru ovog diplomskog rada potrebno je riješiti sljedeće zadatke kako bi se na kraju uspješno validirao algoritam za brzu detekciju i praćenje medicinskog robota RONNA:

1. Treniranje neuronske mreže YOLO v5 sa slikama medicinskog robota RONNA,
2. Dohvaćanje prostora točaka kamere,
3. Razvoj algoritma za detekciju objekta,
4. Implementacija algoritma u ROS-u,
5. Validacija algoritma u laboratoriju.

Kao prvi korak, potrebno je istrenirati neuronsku mrežu YOLO v5 da na slikama detektira RONNA-u. Zatim treba dohvatiti prostor točaka kamere kako bi se iz dvodimenzionalne slike dobila udaljenost traženog objekta od objektiva kamere. Također je potrebno razviti algoritam koji će pomoću YOLO v5 mreže i prostora točaka kamere određivati poziciju objekta u prostoru. Taj algoritam potrebno je implementirati u robotskom operativnom sustavu (ROS) jer će se algoritam pokretati na mobilnom robotu. Na kraju je potrebno razvijeni sustav validirati u laboratoriju.

# 2. STEREOVIZIJSKI SUSTAV INTEL REALSENSE

## 2.1. Općenito o stereovizijskim sustavima

Stereovizijski sustav podrazumijeva sustav od dvije kamere čijom se kombinacijom može odrediti dubina određene točke. Osim dvije kamere, najčešće se u stereovizijskim sustavima nalazi i infracrvena kamera koja ima sposobnost raspoznavanja u mraku.

## 2.2. Pregled tržišta stereovizijskih sustava

### Microsoft Kinect

Microsoft Kinect originalno je služio kao dodatak za igraču konzolu Xbox 360 s ciljem unaprjeđenja korisničkog iskustva igrača i omogućavanja igranja i upravljanja sučeljem samo gestama bez potrebe za joystickom. Prva verzija Kinecta predstavljena je u studenom 2010. godine. Godinu kasnije, Microsoft je izdao beta verziju razvojnog okruženja Kinect for Windows SDK (eng. Software development kit) te je omogućio korištenje izvornih funkcija za izradu Windows aplikacija u programskim jezicima C++, C# (C sharp) i Visual Basic. Kasnije iste godine objavljena je i puna verzija razvojnog okruženja, a tvrtka se pohvalila da surađuje sa stotinama kompanija da im pomogne otkriti što je sve moguće postići s njihovim novim proizvodom. 2012. godine predstavljen novi Kinect namijenjen za Windows platformu i razvoj specijaliziranih aplikacija. 2013. godine zajedno s novom Xbox One igraćom konzolom izašao je novi Kinect za Xbox One koji se prodavao i kao Kinect za Windows v2 pakiran s potrebnim adapterima s USB izlazom namijenjen za korištenje na računalu. Microsoft Kinect v2 prikazan je na slici 5.

**Slika 5. Microsoft Kinect v2 [6]**

**Intel RealSense**

Intel je 2015. godine predstavio svoju liniju kamera *RealSense* namijenjenih za autonomne sustave, mobilne robote, pametne uređaje i ostale slične uređaje. RealSense kamere sastoje se od vizijskog procesora, modula za dubinu i praćenje te dubinskih kamera. Programsko okruženje je otvorenog koda i dostupno je na svim platformama (Windows, Linux, MacOS) te podržava brojne programske jezike (Python, C++, C#, Matlab, ROS i druge).

## 2.3. Intel RealSense D435

Prilikom izrade ovog rada korištena je Intel RealSense D435 dubinska kamera. Kamera je pogodna za korištenje u otvorenim i zatvorenim prostorima, a sastoji se od dvije mono leće i infracrvenog projektora koji po potrebi pomaže pri određivanju dubine u uvjetima slabijeg osvjetljenja. Vidno polje kamere kreće se od 0,3 do 3 metra. S ovom kamerom moguće je odrediti dubinu na rezoluciji do 1280x720 piksela pri 90 FPS. Za potrebe ovog rada koristit će se informacija o dubini određenog piksela što se u RealSense dokumentaciji naziva *Z-Depth*. Točnost određivanja dubine, odnosno apsolutna pogreška, iznosi 2 %, a ovisi o ukupnoj udaljenosti. Podatak o točnosti nalazi se u podatkovnoj tablici kamere. [7] Kamera je prikazana na slici 6.

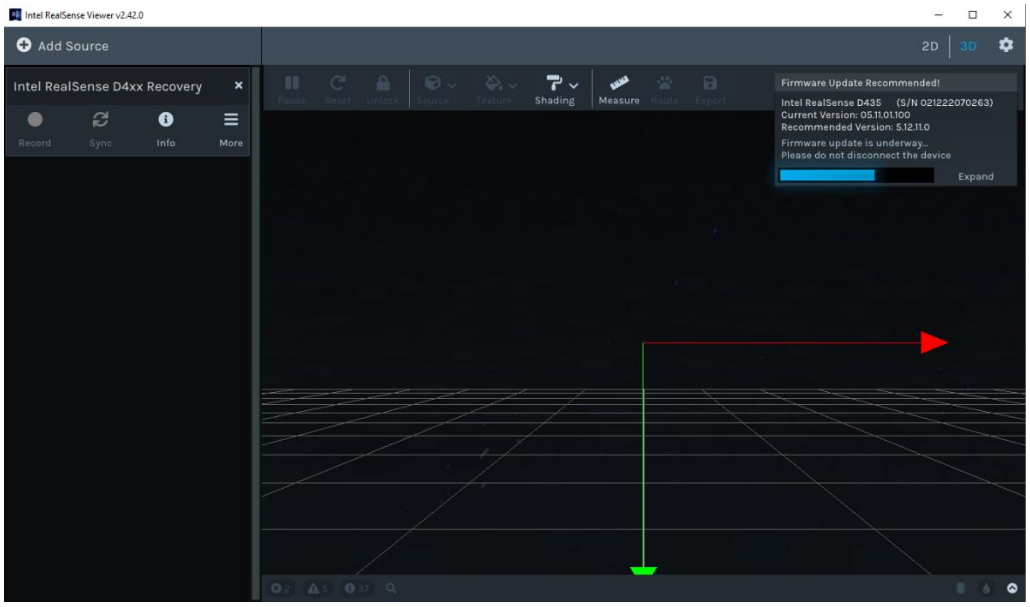
**Slika 6. Intel RealSense D435 [8]**

## 2.4. Postavljanje i instalacija stereovozijskog sustava

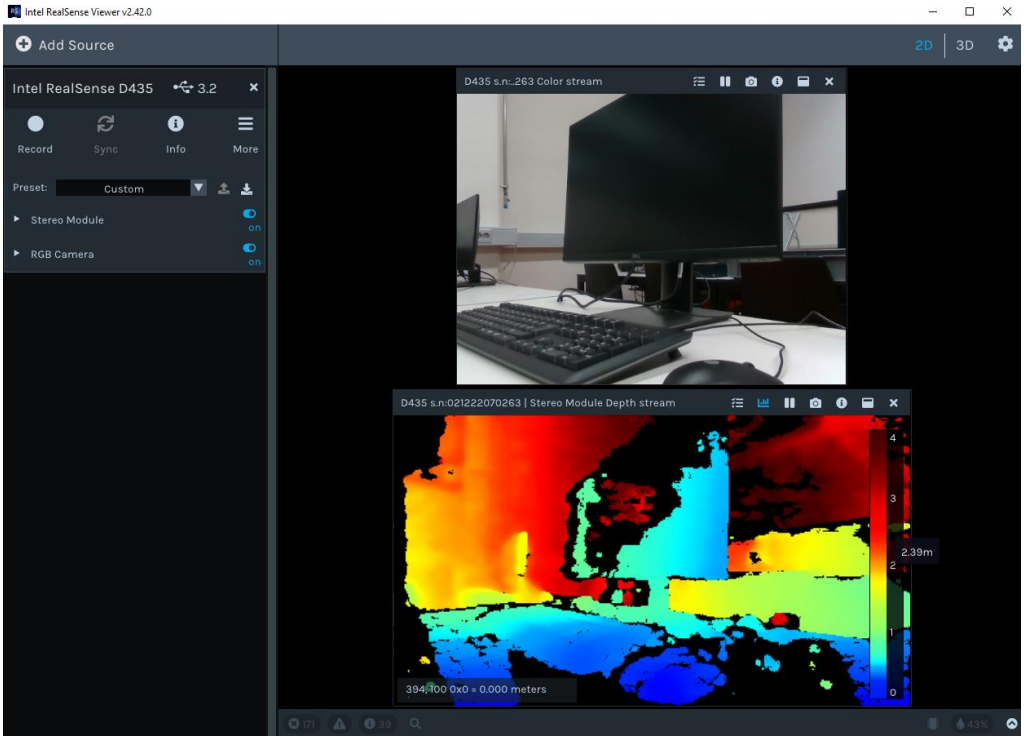
Stereovizijski sustav Intel RealSense D435 dolazi u pakiranju s tronošcem i kabelom za povezivanje na računalo. Sadržaj pakiranja prikazan je na slici 7.


**Slika 7. Sadržaj pakiranja Intel RealSense D435 kamere**

Kabel s jedne strane ima USB konektor koji se spaja na računalo, a s druge strane USB C konektor koji se spaja na kameru. Da bi se kamera mogla koristiti, na računalo je potrebno instalirati Intel RealSense Viewer pomoću kojeg se upravlja kamerom. Unutar sučelja Intel RealSense Viewera moguće je paliti i gasiti stereo modul i RGB kameru, namještati postavke slike, okidati slike, snimati video i namještati razne druge postavke. Na slici 8 prikazano je sučelje Intel RealSense Viewer programa, a na slici 9 prikaz stereo slike i RGB slike s kamere. Na stereo slici bojama su označene različite dubine.



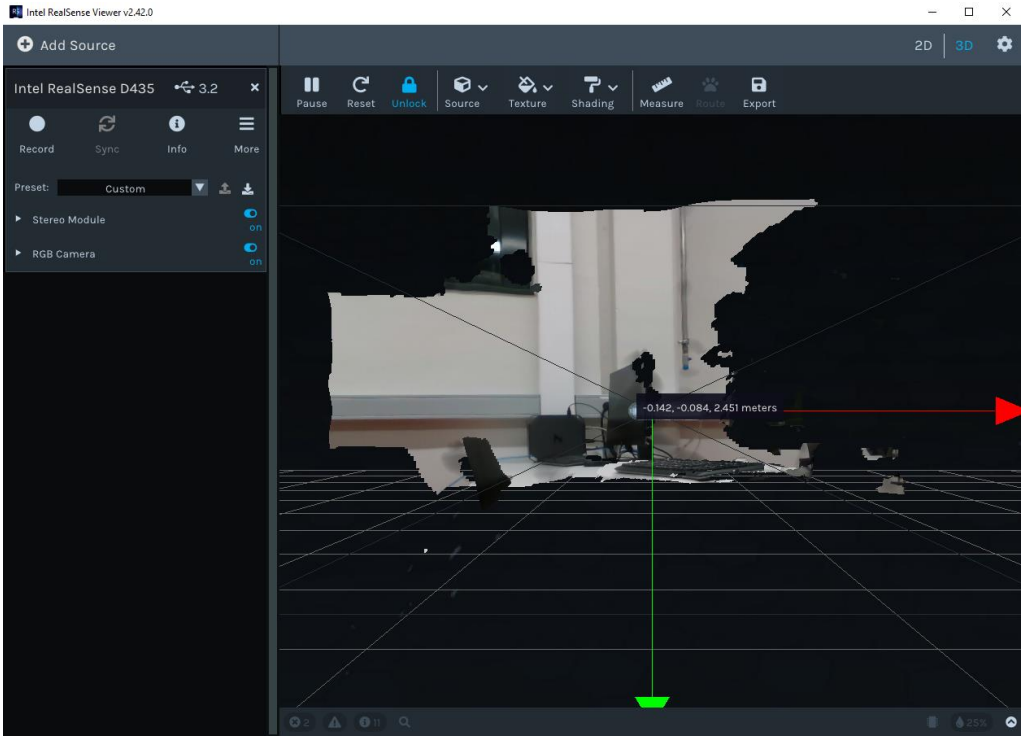

**Slika 8. Sučelje Intel RealSense Viewer programa**




**Slika 9. Prikaz stereo slike i RGB slike s kamere**

Tamno crvenom bojom prikazano je ono što se nalazi najdalje od objektiva kamere, a plavom bojom ono što je najbliže objektivu kamere. Kada se pokazivač miša zadrži u jednoj točki na stereo slici, u donjem lijevom kutu te slike ispisuju se X i Y koordinate u pikselima za tu točku te udaljenost te točke u metrima od objektiva kamere.

Osim prikaza dvodimenzionalnih slikan unutar Intel RealSense Viewera moguće je pregledavati i 3D sliku. Prikaz trodimenzionalne slike nalazi se na slici 10.




**Slika 10. Prikaz trodimenzionalne slike s koordinatama jedne točke**

Dok je dvodimenzionalne slike moguće spremiti u .png formatu, trodimenzionalne se spremaju u .ply formatu odnosno poligonalni format datoteke (eng. Polygon File Format). Radi se o jednostavnom formatu za opisivanje objekta kao poligonalnog modela. Format je izumljen u laboratoriju na Stanfordu 1990. godine gdje se koristio za modele dobivene skeniranjem trodimenzionalnim laserskim triangulacijskim skenerom. [9]

# 3. MODEL ZA DETEKCIJU OBJEKATA YOLO V5

## 3.1. Općenito o algoritmima za detekciju objekata

Ljudski vizualni sustav je toliko brz da ljudi kada pogledaju neku sliku u sekundi znaju tko ili što je na slici, gdje te u kakvoj interakciji su objekti na slici. Takav sustav omogućuje nam da izvodimo vrlo kompleksne radnje s malo donošenja odluka kao što je vožnja auta. Brzi i točni algoritmi za detekciju objekata omogućit će računalima da izvode takve složene radnje bez potrebe za posebnim senzorima i osigurat će potencijal za ostvarivanje interaktivne robotike. Postojeći sustavi za prepoznavanje objekata koriste klasifikatore za svaki objekt kojeg trebaju prepoznati. Određeni klasifikator procjenjuje na različitim mjestima i u različitim mjerilima na slici. Jedna od poznatijih metoda, R-CNN (eng. Region Based Convolutional Neural Networks) tj. konvolucijska neuronska mreža bazirana na regijama, prvo odredi potencijalni okvir oko traženog objekta pa zatim na tom mjestu pomoću klasifikatora određuje stvarne granice objekta. Takav složeni sustav je spor i težak za optimizaciju jer se svaka individualna komponenta mora trenirati zasebno. [10]

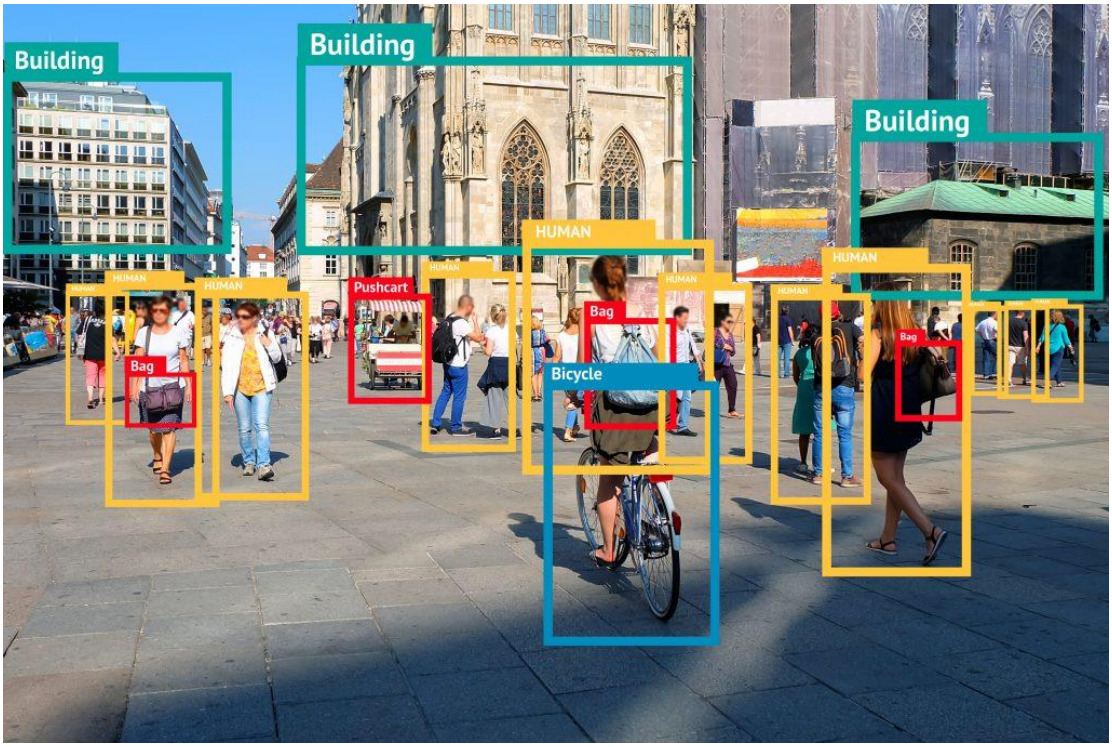
## 3.2. Umjetne neuronske mreže

Pojam umjetna neuronska mreža dolazi od neuronskih mreža živih bića zato što su samoorganizirane i prilagodljive kao i biološke. Neuronske mreže proučavaju se u različitim znanstvenim područjima pa je i njihov broj jako velik. S obzirom na model neurona neuronskih mreža, dijele se na statičke unaprijedne i dinamičke povratne neuronske mreže. [11] U širem smislu, umjetna neuronska mreža definira se kao replika ljudskog mozga koja nastoji simulirati model učenja i obrade podataka. [12] Neke značajke bioloških neuronskih mreža koje ih čine superiornijima u odnosu na najsofisticiranije računalne sustave umjetne inteligencije su:

- Robusnost i tolerancija na pogrešku,
- Fleksibilnost – mreža se sama prilagođava novom okruženju bez nekog preprogramiranja,
- Sposobnost rada u raznim situacijama – mogu se nositi s informacijama koje su nestrukturirane, sadrže šumove i nisu konzistentne,
- Kolektivna obrada podataka – rutinsko paralelno izvođenje operacija. [13]

### 3.3. YOLO model za detekciju objekata

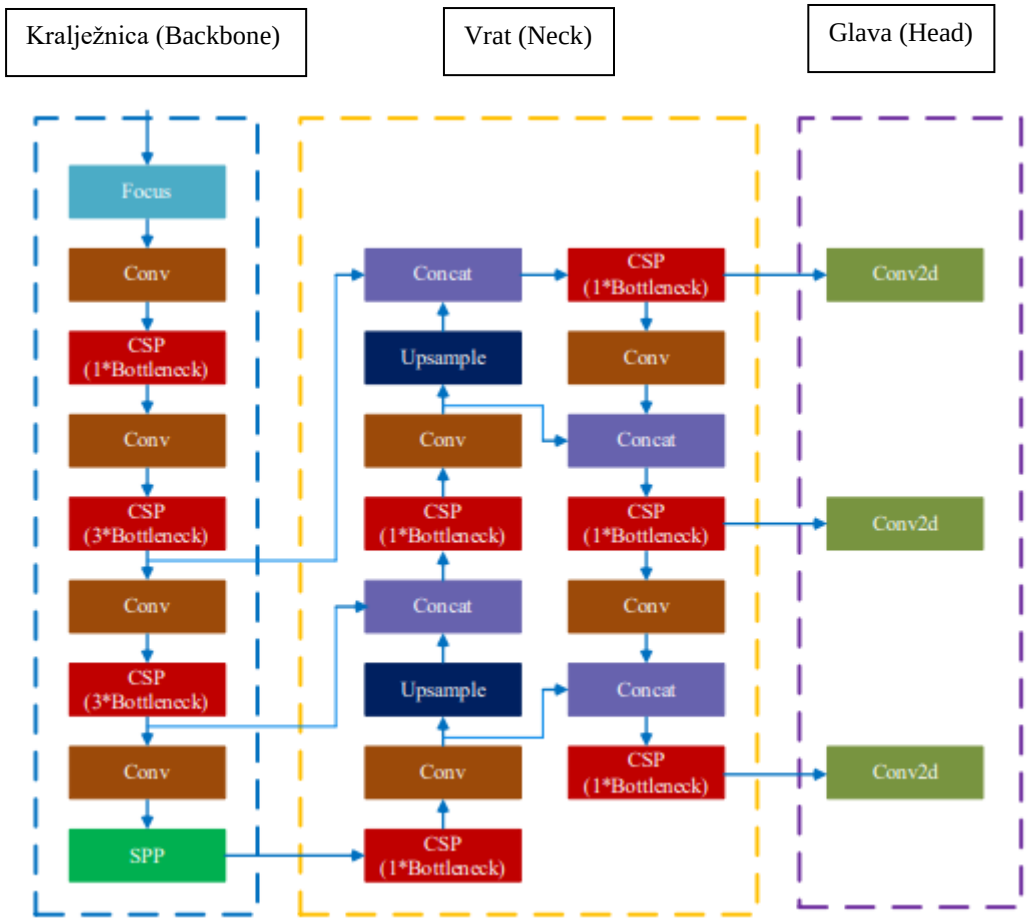
YOLO (eng. You Only Look Once) jedna je od najpopularnijih i najkorištenijih neuronskih mreža za detekciju objekata. Njegova prednost je brzina detekcije i sposobnost da detektira objekte u realnom vremenu. Prva verzija YOLO modela predstavljena je u svibnju 2016. godine. Od tada izdano je ukupno pet verzija, a posljednja verzija koja nosi naziv YOLO v5 predstavljena je u lipnju 2020. godine. Na slici 11 prikazan je primjer detekcije objekata pomoću.




**Slika 11. Primjer detekcije objekata na slici neuronsko mrežom YOLO [14]**

YOLO v5 model sastoji se od tri glavna dijela: kralježnice (eng. Backbone), vrata (eng. Neck) i glave (eng. Head). Prvi dio, kralježnica, izdvaja bitna obilježja dobivene slike. U YOLO v5 modelu sadržan je Darknet, koji kreira CSPDarknet kao svoju kralježnicu. U usporedbi s Darknet53 koji je bio korišten kod YOLO v3 mreže, CSPDarknet je dobio brojna poboljšanja u procesorskoj brzini s istom ili čak većom točnošću. Drugi dio, vrat, zadužen je za kreiranje piramida značajki što pomaže YOLO v5 modelu da identificira slike s istim objektom, ali u drugim dimenzijama i skaliranju. Zadnji dio, glava, generira sidrene okvire (eng. anchor boxes) za mape značajki i izlazne vektore s vjerojatnostima i graničnim okvirima (eng. bounding boxes) za detektirane objekte. [15]

Na slici 12 prikazana je prethodno opisana arhitektura YOLO v5 modela.




**Slika 12. Prikaz arhitekture YOLO v5 modela [15]**

# 4. TRENIRANJE NEURONSKE MREŽE VLASTITIM PODATCIMA

Glavni cilj zadatka je izrada algoritma za detekciju i praćenje medicinskog robota RONNA, a da bi se to postiglo potrebno je istrenirati neuronsku mrežu YOLO v5 sa slikama robota koji je potrebno detektirati. Prvi korak za treniranje neuronske mreže vlastitim slikama je kreiranje baze slika objekta koji se želi naučiti detektirati. Drugi korak je treniranje neuronske mreže s tim slikama.

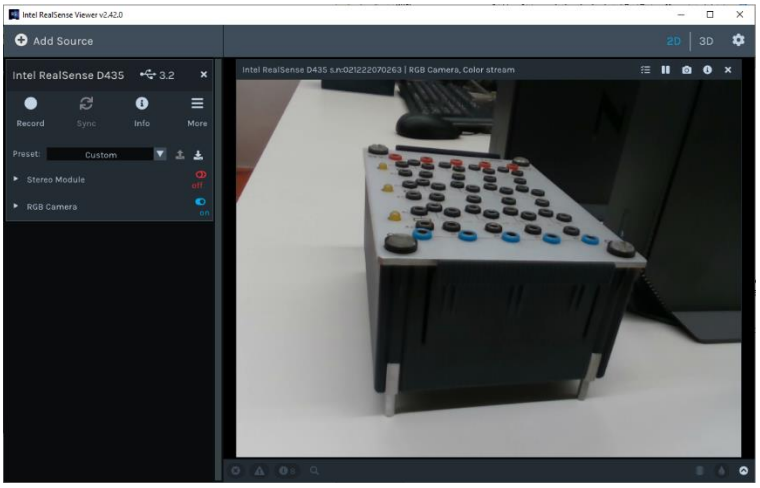
## 4.1. Upoznavanje s radom neuronske mreže YOLO v5

Prije kreiranja baze slika medicinskog robota RONNA, u svrhu upoznavanja rada s neuronskom mrežom YOLO v5, kreirane su baze slika dva manja objekta. Radi se o FESTO razvodnoj kutiji za pneumatiku te bočici vode. Objekti su prikazani na slici 13.


**Slika 13. FESTO razvodna kutija (lijevo) i bočica vode (desno)**

### *4.1.1. Kreiranje baze slika*

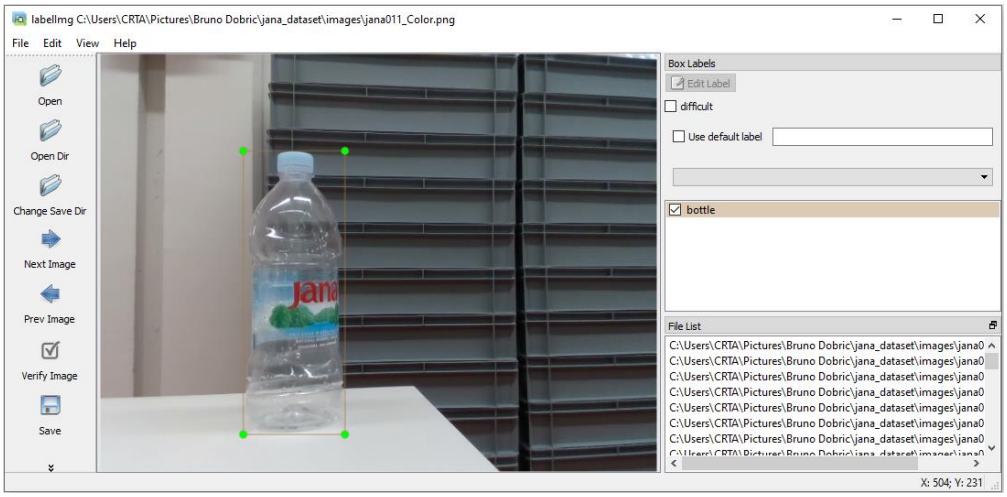
Prilikom kreiranja baze slika potrebno je voditi računa o tome da se objekt slika iz raznih kutova, u raznim okruženjima, pri različitim osvjetljenjima i na način da se u kadru ne vidi cijeli objekt. U ova dva probna slučaja, ukupno je slikano 40 slika svakog objekta. Slike su slikane korištenjem Intel RealSense Viewer sučelja. Na slici 14 prikazana je FESTO razvodna kutija na Intel RealSense Viewer sučelju.




**Slika 14. FESTO razvodna kutija na Intel RealSense Viewer sučelju**

### *4.1.2. Označavanje slika*

Nakon kreiranja baze slika potrebno je na svakoj slici označiti željeni objekt za prepoznavanje. Za označavanje slika postoji više alata, a jedan od njih je labelImg koji je korišten u ovom radu. LabelImg pisan je u pythonu i koristi Qt za grafičko sučelje. Podržani formati oznaka su XML, YOLO i CreateML, a u ovom radu korišten je format XML. Alat je instaliran i pokretan iz Anaconda Prompta. Korištenje labelImg alata je jednostavno i intuitivno. Za početak je potrebno odabrati folder sa slikama te zatim na svakoj slici pravokutnikom označiti traženi objekt kojem se dodjeljuje odgovarajuća oznaka (naziv objekta koji je označen). Na slici 15 prikazano je LabelImg sučelje.




**Slika 15. LabelImg sučelje**

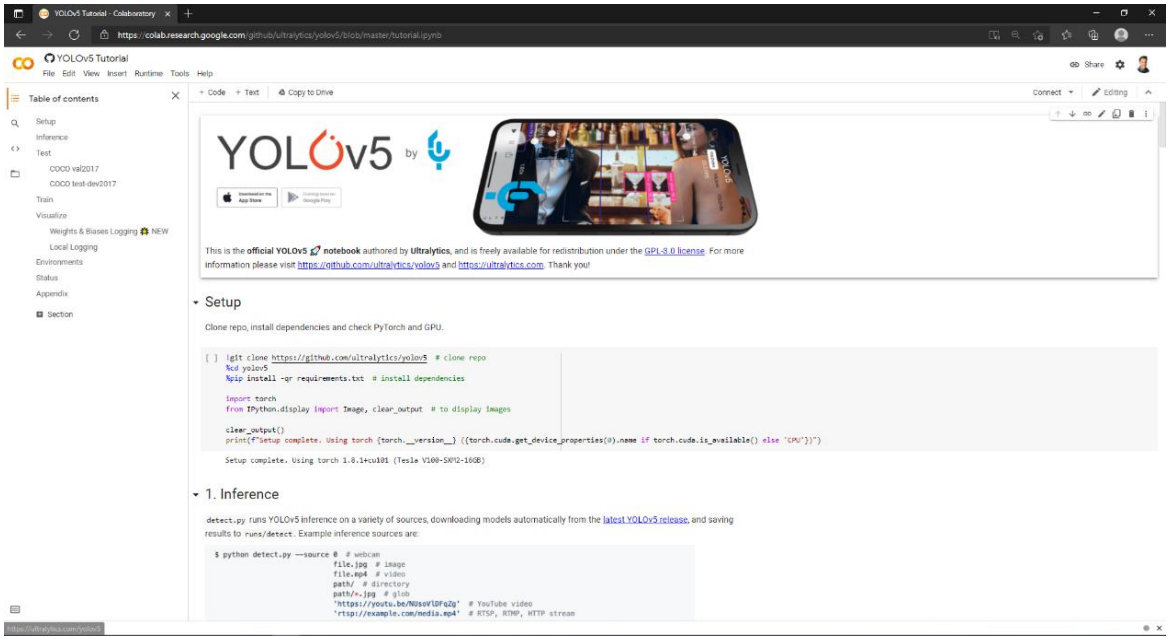
Prilikom označavanja objekta na slikama poželjno se držati nekoliko savjeta koji su se u praksi pokazali točnima. Ipak, iako su ti savjeti generalno točni, treba uzeti u obzir i kontekst svakog pojedinog slučaja. Savjeti su:

1. Označiti pravokutnik oko cijele površine objekta. Bolje je kada se u granični okvir uključi i mali dio prostora oko objekta, nego da dio objekta bude presječen graničnim okvirom.
2. Objekte koji se ne vide u potpunosti na slici potrebno je označiti kao da se vide. U slučaju da je objekt prekriven nekim drugim objektom, bolje je označiti ga kao da se u potpunosti vidi tako da model razumije stvarne granice objekta.
3. Objekte koji su djelomično na slici, tj. ako dio objekta izlazi izvan okvira slike, savjet je da se i oni označe. To ipak ovisi o pojedinom slučaju i koji je krajnji cilj, ali na kraju i dio objekta pripada tom objektu. [16]

Nakon što se na jednoj slici označe potrebni objekti, oznake se spremaju u XML datoteku koja nosi naziv jednak kao slika te se prelazi na sljedeću sliku. Kada se označe sve slike, potrebno je dobiti za svaku sliku tekstualni dokument u kojem su zapisane koordinate oznake objekta na slici. Za to je korišten web alat Roboflow koji radi na način da se preko web sučelja učitaju sve slike i XML datoteke te se odabere format za preuzimanje izlaznih podataka. Odabran je format YOLO v5 PyTorch. Preuzeta zip datoteka sadrži tri foldera (test, train i valid) te yaml datoteku. Zip datoteka se zatim učita na Google Drive radi daljnjeg korištenja.

### *4.1.3. Treniranje neuronske mreže*

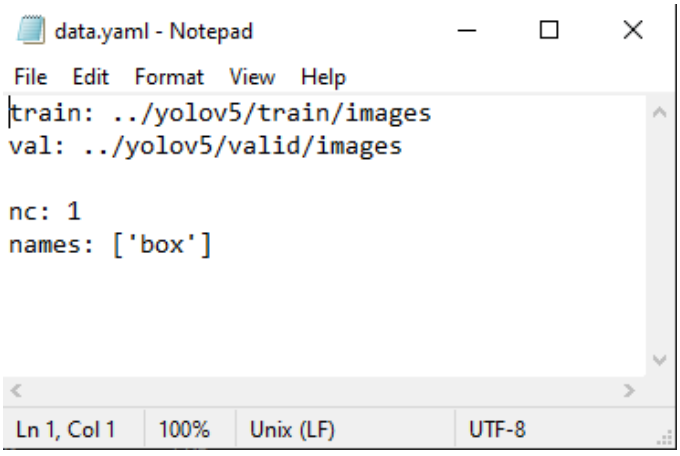
Za treniranje YOLO v5 modela s vlastitom bazom slika korištena je platforma Google Colab. Radi se o programskom okruženju tvrtke Google namijenjenom za programiranje u oblaku. Google Colab omogućuje pisanje i pokretanje python koda, kreiranje bilježnica, spremanje na Google disk te dijeljenje s drugim korisnicima. Colab podržava brojne popularne biblioteke za strojno učenje koje se mogu jednostavno učitati. Još jedna velika prednost Colaba je mogućnost korištenja grafičkog procesora putem internet preglednika i to besplatno. Razlog zašto je Google omogućio korisnicima korištenje grafičkog procesora je taj da nametne svoje alate kao standard u obrazovnim ustanovama kada se radi o strojnom učenju. [17] Sučelje programskog okruženja Google Colab prikazano je na slici 16.




**Slika 16. Sučelje programskog okruženja Google Colab**

Prvi korak u Google Colab okruženju bio je kloniranje repozitorija s Githuba koji sadrži YOLO v5 datoteke te instalacija potrebnih paketa. Sljedeći korak je preuzimanje zip datoteke iz Google diska te raspakiravanje datoteke. Nakon toga može početi proces treniranja modela. Skripta koja služi za treniranje YOLO v5 modela zove se train.py, a argumenti s kojima se poziva su:

- prvi argument je širina slika za treniranje u pikselima koja iznosi 640,
- drugi argument odnosi se na snagu grafičke kartice računala na kojem se vrti skripta,
- treći argument, odnosno broj epoha, ključan je za kvalitetno treniranje mreže,
- četvrti argument je konfiguracijska yaml datoteka koja sadrži putanje do mape sa slikama i s oznakama te naziv objekata koji su označeni na slikama. Sadržaj konfiguracijske datoteke prikazan je na slici 17.




**Slika 17. Sadržaj konfiguracijske yaml datoteke**

- Peti argument je putanja do datoteke s težinama s kojima se trenira mreža. Kada se prvi put trenira mreža vlastitim podatcima, kao datoteka s težinama koristi se zadana datoteka YOLO v5 mreže.

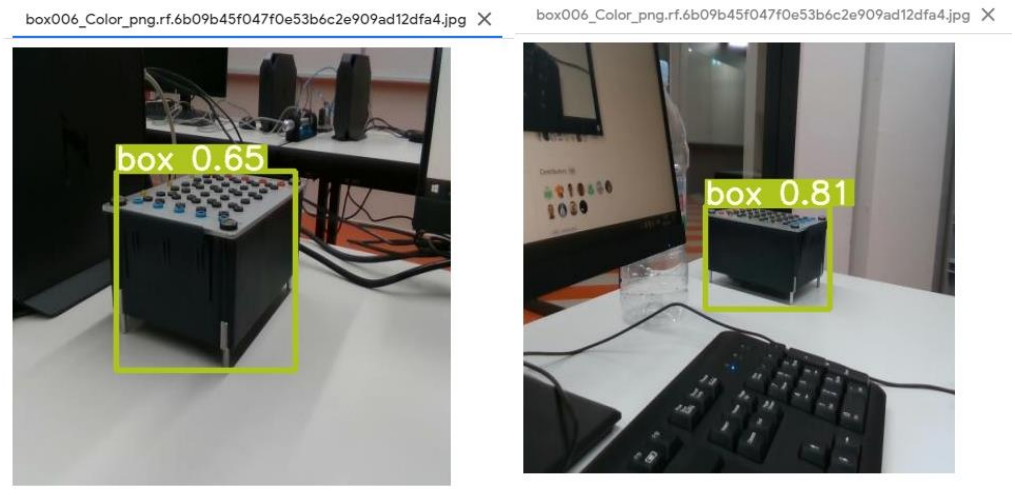
Skripta train.py pozvana je s potrebnim argumentima iz Anaconda Prompta.

Rezultati skripte za treniranje su datoteke last.pt i best.pt koje sadrže težine. Datoteka best.pt sadrži težine koje daju najbolje performanse mreže, a last.pt sadrži težine dobivene nakon zadnje epohe te se može koristiti za nastavak treniranja. Prepoznavanje objekata na slikama izvodi se pomoću python skripte detect.py koja se poziva s argumentima:

- ulaz (source) koji može biti mapa sa slikama, video, stream ili prijenos s web kamere,
- argument težine (weights) koji podrazumijeva putanju do datoteke s težinama,
- željena podudarnost (conf).

Rezultati detekcije objekta pomoću skripte detect.py su slike na kojima su ucrtani granični okviri oko detektiranih objekata iznad kojih piše vjerojatnost da se radi o traženom objektu.

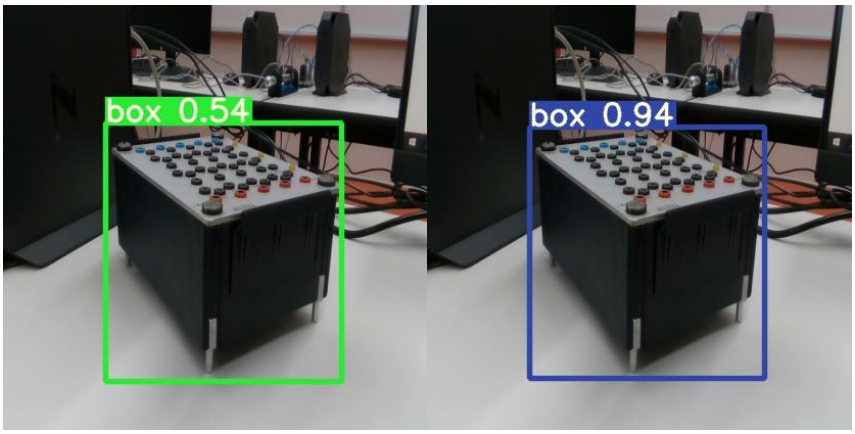
Mreža je trenirana s različitim brojem epoha, a rezultati su pokazali da broj epoha utječe na kvalitetu treniranja modela, ali utječe i na vrijeme treniranja. U ovom slučaju isprobano je treniranje s različitim brojem epoha i rezultati su pokazali da što je veći broj epoha model je sigurniji da se radi o objektu kojeg je trebao prepoznati. Kada se treniralo s malim brojem epoha, na primjer manjim od 50, mreža nije uspješno prepoznala ni jedan objekt na slikama. Nakon treniranja s 80 epoha prepoznat je objekt sa 65 % i s 81 % vjerojatnošću, u uvjetima kada je dobro vidljiv na slici. Rezultati detekcije objekta prikazani su na slici 18.



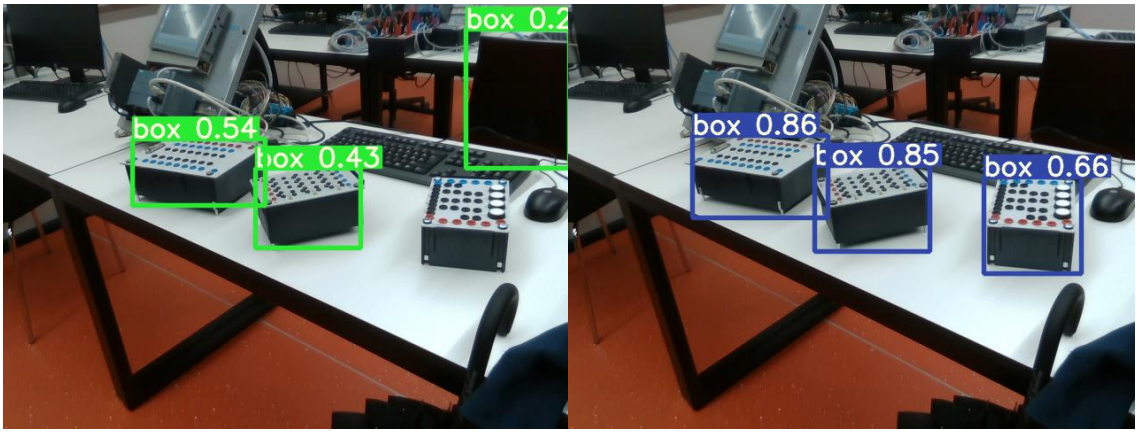

**Slika 18. Rezultati prepoznavanja objekta nakon treniranja s 80 epoha**

Kada se utvrdilo da treniranje mreže s većim brojem epoha za rezultat ima bolje rezultate, mreža je trenirana s još većim brojem epoha. Prvo je mreža trenirana sa 120 epoha i to je dalo rezultate

prepoznavanja do oko 80 % sigurnosti što se odnosi na jasne slike na kojima je dobro vidi objekt prepoznavanja, a za slike gdje je objekt djelomično skriven, u daljini ili drugačijoj poziciji ta sigurnost je manja. Nakon toga cilj je bio treniranje modela s puno većim brojem epoha te usporedba dobivenih rezultata. Odabrani broj epoha bio je 500. Budući da treniranje dugo traje, provedeno je lokalno na računalu koristeći Anaconda Prompt, umjesto u programskom okruženju Google Colab. Rezultati su pokazali da je s težinama koje se dobiju nakon treniranja modela s 500 epoha sigurnost prepoznavanja objekta na slici veća od 90 % za bolje slike, a za slike na kojima se objekt vidi u daljini sigurnost prepoznavanja je iznad 50 %. Usporedba rezultata detekcije mreže trenirane sa 130 i 500 epoha prikazana je na slikama 19 i 20.




**Slika 19. Usporedba rezultata detekcije objekta nakon treniranja mreže sa 130 (lijevo) i 500 epoha (desno)**




**Slika 20. Usporedba rezultata detekcije objekta nakon treniranja mreže sa 130 (lijevo) i 500 epoha (desno)**

Iz dobivenih rezultata uočljivo je da broj epoha ima važnu ulogu u treniranju mreže. Kada se mreža trenira s manjim brojem epoha postoje slučajevi krivog prepoznavanja objekta, tj. da se kao traženi objekt prepozna neki drugi objekt. S druge strane, kod mreže trenirane s velikim brojem epoha nije se javio slučaj krivog prepoznavanja, ali je u slučaju FESTO razvodnih kutija

kao objekt prepoznata čak i kutija koja ima malo drugačiju gornju stranu od one s kojom je mreža trenirana.

## 4.2. Treniranje neuronske mreže slikama medicinskog robota RONNA

Nakon upoznavanja s principom rada neuronske mreže YOLO v5, sljedeći korak je treniranje mreže slikama medicinskog robota RONNA čija detekcija je tema ovog rada. Budući da RONNA ne izgleda sa svih strana jednako, prilikom kreiranja baze slika bit će potrebno slikati ju sa svih strana.

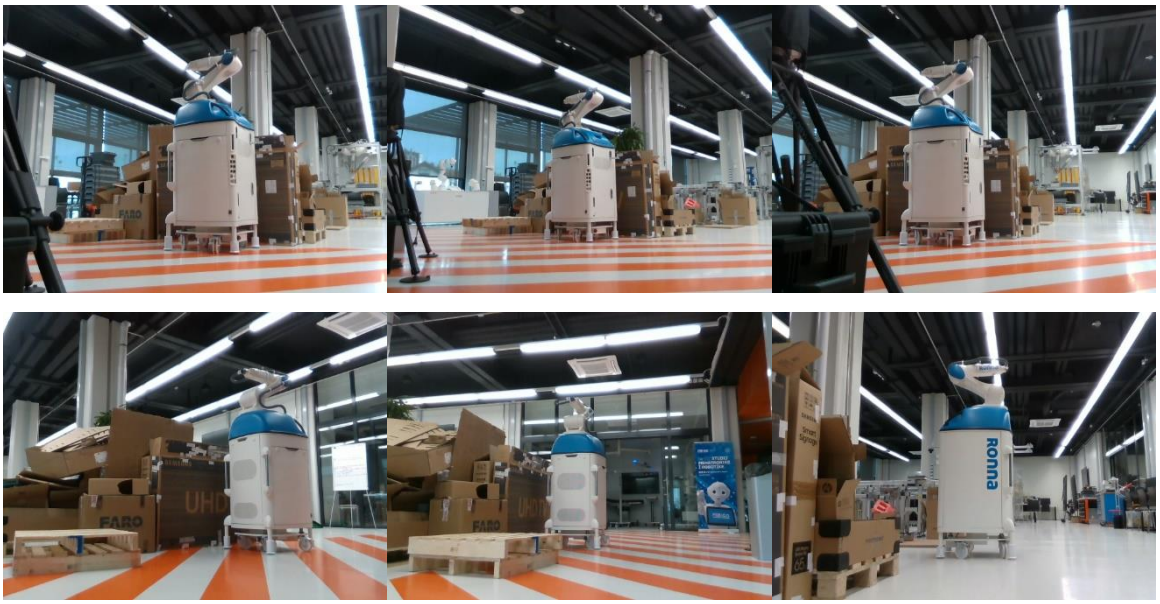
### *4.2.1. Kreiranje baze slika medicinskog robota RONNA*

Za potrebe kreiranja baze slika medicinskog robota RONNA korišten je postav koji se sastoji od Intel RealSense D435 kamere koja je spojena na prijenosno računalo, prijenosnog računala na kojem je pokrenut Intel RealSense Viewer i transportnog podloška s kotačićima za Euro kutije. Visina postava i udaljenost kamere od poda približno je slična visini na kojoj bi bila kamera na mobilnom robotu za prijevoz RONNA-e. Slika 21 prikazuje postav za slikanje medicinskog robota RONNA, a slika 22 prikazuje samo slikanje.

**Slika 21. Postav za slikanje medicinskog robota RONNA**

Robot RONNA slikan je 120 puta pomoću prethodno opisanog postava iz različitih kutova i perspektiva tako da se pomicao i robot i postav. Za razliku od FESTO razvodne kutije koja je slikana 40 puta, robot RONNA slikan je 120 puta zato što ima kompleksniju geometriju i različito izgleda sa svih strana. Osim toga, cilj zadatka je izrada algoritma za prepoznavanje robota pa je važno odmah pri kreiranju baze slika voditi računa da te slike budu što sličnije onima koje će slikati kamera na mobilnom robotu. Neke od slika koje su se koristile kao baza za treniranje mreže prikazane su na slici 23, a neke od slika s okluzijom na slici 24.

**Slika 22. Prikaz slikanja medicinskog robota RONNA**

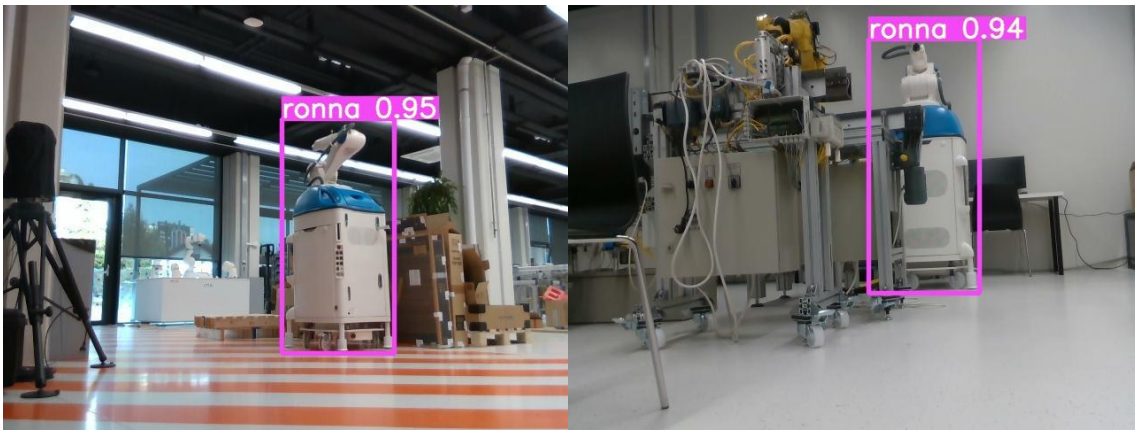



**Slika 23. Prikaz slika RONNA-e**


**Slika 24. Prikaz slika RONNA-e s okluzijama**

### *4.2.2. Treniranje neuronske mreže slikama robota RONNA*

Mreža je trenirana sa 120 slika medicinskog robota RONNA sa 600 epoha. Prilikom treniranja neuronske mreže korištena je HP radna stanica. Treniranje je trajalo 9,262 sata. Rezultat treniranja sa 600 epoha je prepoznavanje robota sa sigurnošću prepoznavanja većom od 90 %. Rezultati detekcije prikazani su na slici 25. Iz prikazanih rezultata vidljivo je da je mreža jako dobro istrenirana te da čak prepoznaje željeni objekt kada je djelomično prekriven.




**Slika 25. Rezultati detekcije robota RONNA mrežom istreniranom sa 600 epoha**

Sljedeći korak je slikanje RONNA-e iz različitih udaljenosti, različitih kutova te različitih perspektiva. Perspektive su: s prednje strane, sa stražnje strane, s bočne strane (pritom je svejedno radi li se o lijevoj ili desnoj strani) te pod 45 stupnjeva. Udaljenosti s kojeg je objekt slikan su od 3 do 10 metara. Rezultati pokazuju da je mreža istrenirana sa 600 epoha dovoljno dobra da prepozna traženi objekt iz različitih perspektiva i udaljenosti s visokom sigurnošću, ali ne i u uvjetima slabog osvjetljenja kada je sigurnost da je prepoznat pravi objekt malo iznad 50 %. Opisani slučaj prikazan je na slici 26.


**Slika 26. Mala sigurnost detekcije objekta u uvjetima slabijeg osvjetljenja**

Ipak, kada se uzmu u obzir okolnosti to je i dalje dobar rezultat. Važno je napomenuti da se nije javio slučaj pogrešne detekcije, tj. da je neki drugi objekt detektiran kao RONNA.

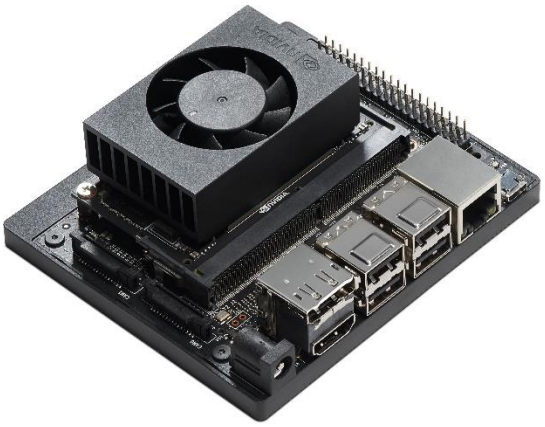
Kada je zaključeno da je mreža trenirana sa 600 epoha zadovoljavajuća u slučaju prepoznavanja robota RONNA, sljedeći je cilj trenirati mrežu s još većim brojem epoha i usporediti dobivene rezultate. Mreža je još trenirana s 1500 i 2000 epoha. Suprotno očekivanjima, dobiveni rezultati nakon tako velikog broja epoha pokazali su se lošijima nego oni sa 600 epoha. Na slici 27 prikazana je usporedba rezultata detekcije na istoj slici s mrežama treniranim sa 600, 1500 i 2000 epoha. Razlika nije jako velika, ali vidljivo je da s povećanjem broja epoha opada točnost detekcije.


**Slika 27. Usporedni prikaz rezultata detekcije s mrežama istreniranim sa 600 epoha (lijevo), 1500 epoha (sredina) i 2000 epoha (desno)**

Razlog tome mogao bi biti taj što je objekt uvijek isti i kod treniranja s više epoha dolazi do pretrenirane mreže. Na primjer, ako se mreža trenira da prepoznaje ljude, aute, pse i slično, tada treba jako puno epoha jer nisu svi ljudi, auti i psi isti, a u ovom slučaju medicinski robot je uvijek isti. Iz tog razloga za treniranje mreže za prepoznavanje medicinskog robota ne treba jako veliki broj epoha, ali je bitno pokriti što više mogućih perspektiva, kutova i slično.

# 5. RAČUNALO NVIDIA JETSON XAVIER NX

Nvidia Jetson Xavier NX je malo super računalo namijenjeno za ugrađene sustave (eng. embedded systems) te pogodno za brzu obradu velikog broja podataka kao što su algoritmi umjetne inteligencije. Računalo je prikazano na slici 28.



**Slika 28. Računalo Nvidia Jetson Xavier NX [18]**

Xavier NX računalo sastoji se od grafičkog procesora, centralnog procesora, ventilatora, četiri USB-A ulaza, Display Port ulaza, HDMI ulaza, Ethernet ulaza, Micro USB ulaza i ulaza za napajanje. Specifikacije računala dane su u tablici 1.

**Tablica 1. Specifikacije računala Nvidia Jetson Xavier NX**

| | |
|---|---|
| Grafički procesor | NVIDIA Volta |
| Središnji procesor | 6 jezgri NVIDIA Carmel ARM®v8.2 64-bit CPU |
| Radna memorija | 8 GB 128-bit LPDDR4x @ 1600 MHz<br>51.2GB/s |
| Pohrana | microSD kartica |
| Povezivost | Gigabit Ethernet |
| Video izlaz | HDMI, Display Port |
| USB ulazi | 4 x USB 3.1, USB 2.0 Micro-B |
| Dimenzije | 103 mm x 90,50 mm x 34,66 mm |

## 5.1. Primjena računala Nvidia Jetson

Jetson Xavier NX spada u skupinu Nvidia Jetson serije malih snažnih računala koji su sposobni pokretati zahtjevne proračune u realnom vremenu, a ujedno su energetski učinkoviti. Nvidia Jetson računala koriste se za pokretanje aplikacija dubokog učenja, upravljanje paralelno izvođenih neuronskih mreža ili za obradu podata visoke rezolucije sa senzora u realnom vremenu. Jetson pločice postale su mjerilo kvalitete za autonomne projekte i prisvojile su ih vodeće tvrtke u svijetu industrijske robotike kao što su FANUC ili Komatsu. [19]

Postoje četiri različita Jetson računala, a to su:

- Nvidia Jetson Nano – najmanji i najpristupačniji, ali i najslabiji po performansama,
- Nvidia Jetson TX2 – izrazito lagan i pogodan za dronove,
- Nvidia Jetson Xavier NX – najmanje super računalo na svijetu, 10 puta jače performanse od TX2,
- Nvidia Jetson Xavier AGX – prvo računalo posebno projektirano za autonomne sustave.

Navedena računala prikazana su na slici 29.

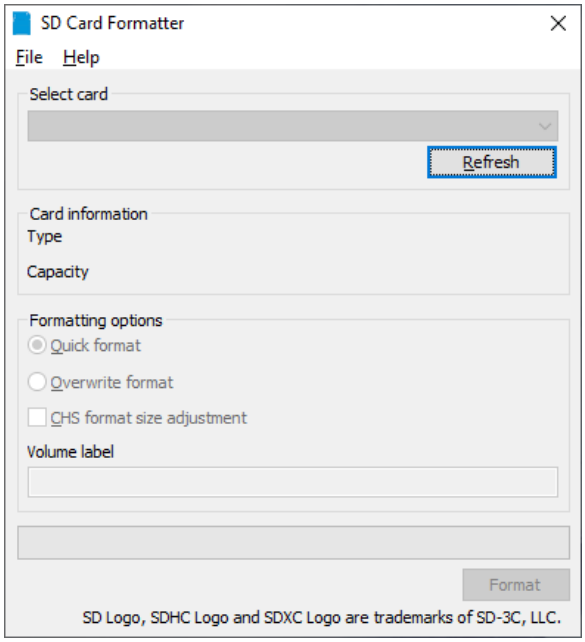
**Slika 29. Nvidia Jetson serija redom: Xavier AGX, Xavier NX, TX2 i Nano [20]**

Prema riječima proizvođača, tvrtke Nvidia, Jetson Xavier NX savršen je za sustave s umjetnom inteligencijom visokih performansi kao što su komercijalni roboti, medicinski instrumenti,

pametne kamere, senzori visoke rezolucije, automatizirana optička testiranja, pametne tvornice i ostale ugradbene pametne sustave. [21]

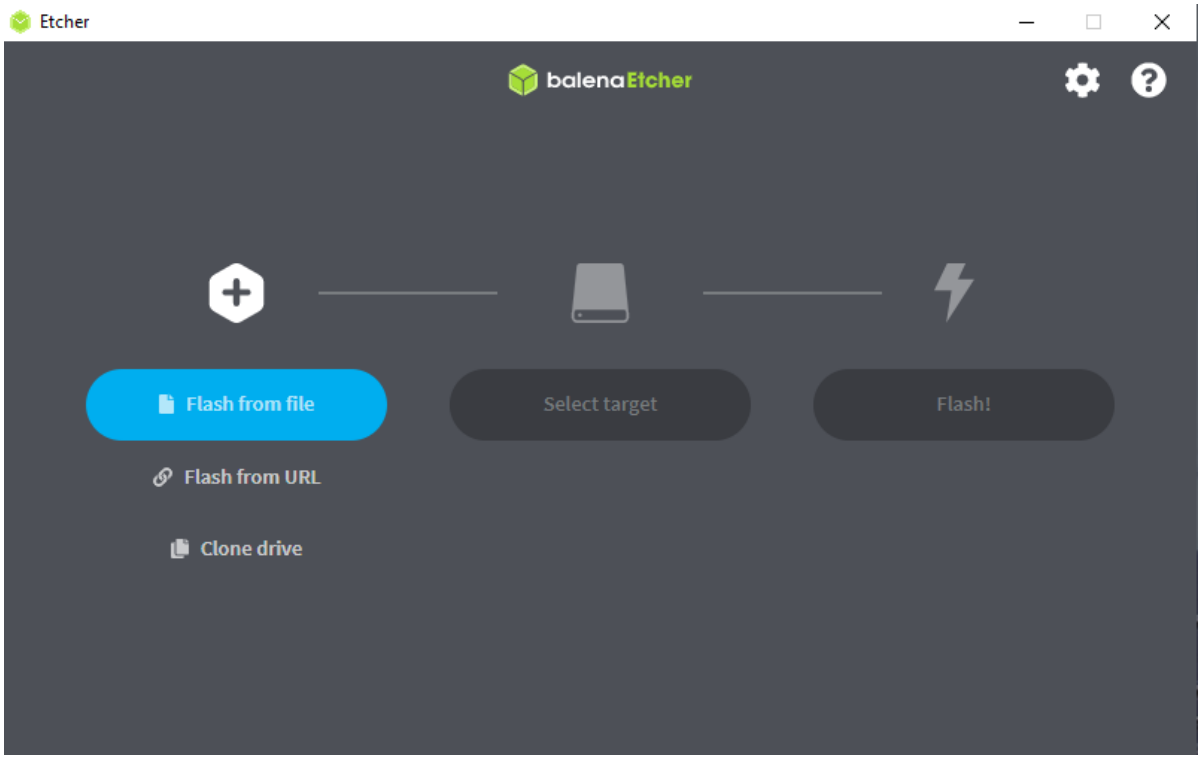
## 5.2. Postavljanje i instalacija računala Nvidia Jetson Xavier NX

Računalo Nvidia Jetson Xavier NX u pakiranju dolazi s kabelom za napajanje. Za instalaciju Linux operativnog sustava na Xavier NX potrebna je SD kartica. Karticu je prvo potrebno formatirati na drugom računalu. Za to je korišten program SD Card Formatter. Na slici 30 prikazano je sučelje programa SD Card Formatter.




**Slika 30. Sučelje programa SD Card Formatter**

Nakon formatiranja kartice, potrebno je preuzeti sliku diska za Xavier NX računalo s Nvidia internet stranice. Također, potrebno je instalirati program Balena Etcher pomoću kojeg se slika diska učita na karticu. Sučelje programa Balena Etcher prikazano je na slici 31. Korištenje programa je jednostavno, sastoji se od tri koraka. Prvi korak je odabrati preuzetu sliku diska, drugi korak je odabrati SD karticu na koju se učita slika diska i treći korak je samo učitavanje slike diska na SD karticu. Taj proces je gotov kroz 10 minuta i može se nastaviti sa sljedećim korakom.




**Slika 31. Sučelje programa Balena Etcher**

Sljedeći korak je umetanje kartice u Jetson Xavier NX te spajanje na potrebnu periferiju i na napajanje. Od periferije potrebni su monitor, tipkovnica i miš. Kada se uključi napajanje, odmah se sustav počinje podizati. Kroz nekoliko minuta sustav se podigne i spreman je za korištenje. [22] Na računalu Xavier NX instaliran je Linux Ubuntu 18.04.

## 5.3. Pokretanje YOLO v5 mreže na Xavier NX platformi

Nakon što je računalo spremno za korištenje potrebno je otvoriti Terminal prozor koji služi za pokretanje i izvršavanje naredbi i skripti. Kao zadani predinstalirani Python na Ubuntu sustavu dolaze Python 2.7 i Python 3.6. Da bi se mogla koristiti YOLO v5 mreža potrebno je instalirati Python 3.8. te pojedine biblioteke kao što su OpenCV i PyTorch. Python omogućuje kreiranje virtualnih okruženja za svaki projekt što je posebno korisno u slučaju da se na istom računalu radi u više verzija Pythona, bilo zbog drugačijih zahtjeva pojedinih projekata, bilo zbog testiranja različitih verzija i biblioteka bez opasnosti da se ugrozi glavna instalacija. [23]

Za aktivaciju virtualnog okruženja potrebno je prvo imati instaliran pip. To se može provjeriti upisivanjem sljedeće linije u Terminal:

```
pip -h
```

U slučaju da pip nije instaliran, potrebno ga je instalirati. Nakon toga, preduvjet za aktivaciju virtualnog okruženja je biblioteka virtualenv. Ta biblioteka instalira se putem pipa.

Nakon instalacije biblioteke virtualenv potrebno je kreirati virtualno okruženje. Za to je potrebno odrediti putanju, tj. folder u kojem će se nalaziti to virtualno okruženje. Folder se

```
pip install virtualenv
```

kreira u aktivnom direktoriju.

```
virtualenv app01
```

Kada je kreiran folder, zadnji korak je aktivacija virtualnog okruženja naredbom source i putanjom do skripte activate. [23]

```
source app01/bin/activate
```

Nakon što je kreirano virtualno okruženje potrebno je instalirati Python 3.8 verziju jer zadana verzija 3.6 ne podržava YOLO v5.

Za instalaciju Pythona 3.8 prvo je potrebno ažurirati liste paketa i predzahtjeve za instalaciju:

```
$ sudo apt update
$ sudo apt install software-properties-common
```

Nakon toga potrebno je dodati deadsnakes PPA na sistemsku listu izvora te potvrditi s tipkom Enter:

```
$ sudo add-apt-repository ppa:deadsnakes/ppa
```

Jednom kada je repozitorij omogućen, može se instalirati Python 3.8:

```
$ sudo apt install python3.8
```

```
$ python3.8 --version
```

Za provjeru da je instalirana dobra verzija Pythona može se utipkati:

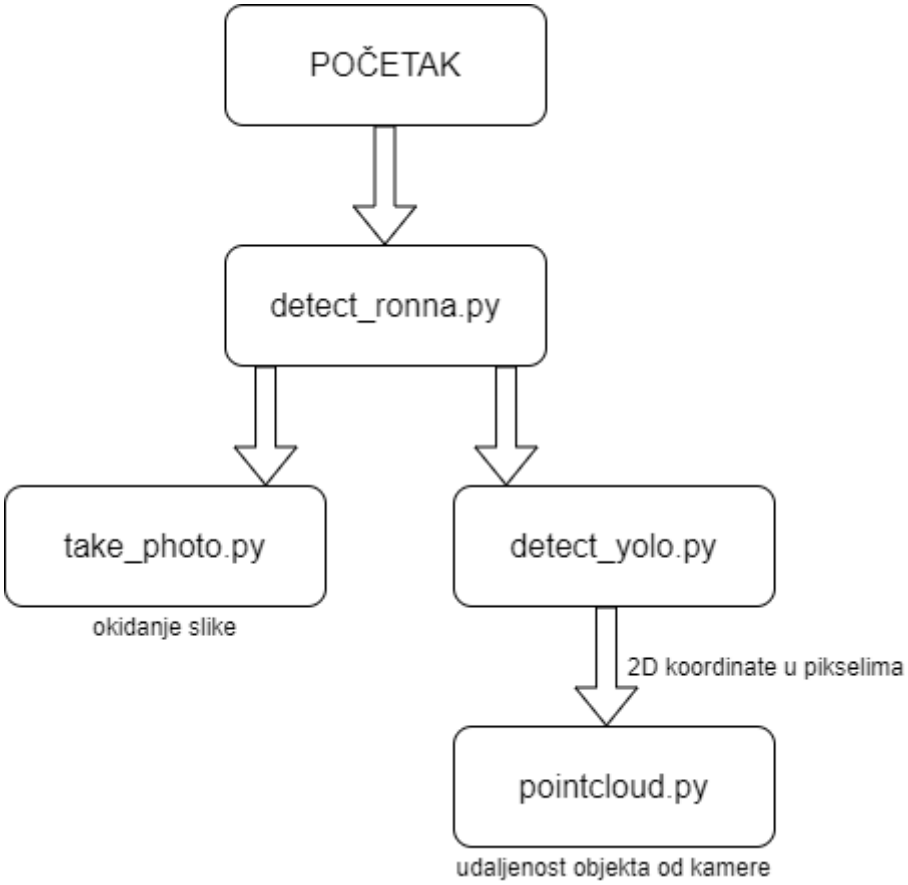
Kada je Python 3.8 uspješno instaliran može se početi s radom. [24]

Nakon instalacije potrebne verzije Pythona, potrebno je instalirati još neke pakete kako bi se uspješno pokrenula YOLO v5 mreža. To su paketi *pyrealsense2, opencv-python, torch* i

*torchvision*. Nakon instalacije navedenih paketa moguće je pokretati treniranje YOLO v5 mreže jednako kao na Google Colab okruženju ili u Anaconda Promptu na Windows računalima.

# 6. ALGORITAM ZA DETEKCIJU I PRAĆENJE

Nakon što je kreiran model za prepoznavanje medicinskog robota RONNA, sljedeći je zadatak izraditi algoritam koji će u realnom vremenu detektirati i pratiti željeni objekt. Ideja algoritma je da u beskonačnoj petlji pokreće funkciju koja okida sliku, zatim funkciju koja detektira objekt na toj slici i konačno funkciju koja ispisuje koordinate težišta objekta u prostoru. Na slici 32 prikazan je dijagram toka izvođenja skripti algoritma za detekciju i praćenje.




**Slika 32. Dijagram toka izvođenje skripti algoritma za detekciju i praćenje**

## 6.1. Izvedba algoritma za detekciju i praćenje

Algoritam za detekciju i praćenje izveden je pomoću četiri python skripte. To su skripte: *detect_ronna.py*, *take_photo.py*, *detect_yolo.py* i *pointcloud.py*. Glavna skripta koja se pokreće je *detect_ronna.py* koja poziva skripte *take_photo.py* i *detect_yolo.py*. Skripta take_photo.py okida sliku pomoću Intel RealSense D435 kamere te ju sprema u zadani folder. Nakon toga poziva se skripta *detect_yolo.py* koja je ustvari modificirana verzija originalne *detect.py* skripte iz YOLO v5 paketa. Kada se pozove, ona uzima spremljenu sliku kao ulazni argument te na njoj vrši detekciju željenog objekta. Podatci o koordinatama graničnog okvira koriste se za

određivanje težišta objekta na slici, odnosno za dobivanje X i Y koordinata težišta. Dobivene koordinate težišta koriste se kao ulazni argument za funkciju koja određuje dubinu točke. Kao rezultat, u terminalu se ispisuju X, Y i Z koordinate težišta detektiranog objekta te informacija da je objekt detektiran. U nastavku će biti detaljno objašnjena svaka pojedina skripta.

### *6.1.1. Skripta detect_ronna.py*

Skripta *detect_ronna.py* glavna je skripta algoritma za detekciju i praćenje medicinskog robota RONNA, prva se poziva i unutar nje se pozivaju druge funkcije. Sastoji se od jedne glavne funkcije u kojoj se nalazi beskonačna *while* petlja unutar koje se pozivaju funkcije za okidanje slika te za detekciju objekta. Na početku skripte učitane su potrebne biblioteke, a to su os i shutil za manipulaciju s folderima te skripte koje se pozivaju. Prije pozivanja funkcije za okidanje slika definirana je putanja do foldera u koji se spremaju slike te putanja do YOLO v5 modela koji sadrži težine za detekciju. Nakon toga se pomoću funkcije *os.path.exist()* provjerava postoji li već folder za spremanje slika, a u slučaju da postoji isti se briše funkcijom *shutil.rmtree()* te se zatim kreira novi prazni folder pomoću funkcije *os.makedirs()*. Nakon kreiranja praznog foldera poziva se funkcija za okidanje slika koja sprema slike u taj folder. Kada se slika spremi u folder odmah se poziva funkcija za detekciju objekta pomoću istrenirane neuronske mreže YOLO v5.

### *6.1.2. Skripta take_photo.py*

Skripta *take_photo.py* poziva se iz glavne funkcije skripte *detect_ronna.py*, a služi za okidanje slika kamerom Intel RealSense D435. Na početku skripte učitavaju se potrebne biblioteke, a to su *pyrealsense2* za manipulaciju s kamerom, *numpy* za korištenje nizova, *cv2* za spremanje slike, *time* za definiranje imena slike prema trenutnom vremenu i *os* za definiranje putanje slike koja se sprema.

Skripta se sastoji od glavne funkcije čiji ulazni argument je putanja do foldera u koji se sprema slika. Prvo se definira *pipeline* i konfiguracija, a zatim se omogućuje *stream* i pokreće *pipeline*. Budući da korištena kamera ima više različitih načina rada, bira se dvodimenzionalna slika u boji. Naziv slike koja se sprema definiran je funkcijom *strftime* iz biblioteke *time* tako da sadrži godinu, mjesec, dan, sat, minutu i sekundu trenutka u kojem je slikana. Nakon toga, pomoću funkcije *imwrite* iz biblioteke *cv2* sprema se slika u zadani folder te se *pipeline* zaustavlja.

### *6.1.3. Skripta detect_yolo.py*

Skripta *detect_yolo.py* modificirana je verzija skripte detect.py koja je sastavni dio YOLO v5 paketa. Skripta je modificirana na način da se može pozivati iz druge skripte s argumentima, umjesto pozivanja iz terminala kako je u originalu. Također, pojedine opcije koje je korisnik mogao prilikom pozivanja skripte odabrati sada su postavljene kao zadane jer za ovu primjenu nisu potrebne. Radi se o spremanju koordinata u tekstualnu datoteku, spremanja odrezanog dijela slike u kojem je detektiran objekt i slično. Osim toga, uz funkciju *detect()* koja služi za detektiranje objekta, dodana je funkcija *get_z()* koja služi za dobivanje udaljenosti težišta detektiranog objekta od kamere, tj. Z koordinatu težišta. Ta koordinata dobije se pomoću funkcije *get_point_cloud()*, a za ulazne argumente uzima X i Y koordinatu točke s dvodimenzionalne slike za koju se treba odrediti dubina. Koordinate X i Y funkciji *get_point_cloud()* prosljeđuju se iz funkcije *detect.py* gdje su izračunate radi ocrtavanja graničnog okvira oko detektiranog objekta. Budući da su u funkciji *detect.py* koordinate zapisane u jediničnom koordinatnom sustavu, potrebno ih je preračunati u koordinatni sustav kamere za prostor točaka koji je određen pikselima, tj. dimenzijom slike 424x240. Kada se dobije Z koordinata, na terminal se ispisuju sve tri koordinate težišta detektiranog objekta.

### *6.1.4. Skripta pointcloud.py*

Skripta *pointcloud.py* na početku učitava samo biblioteku pyrealsense2. Sastoji se od jedne funkcije pod nazivom *get_point_cloud()* koja se poziva iz skripte *detect_yolo()* s koordinatama točke za koje se traži treća dimenzija kao ulaznim argumentima. Prvo se definiraju *pipeline* i konfiguracija pa se zatim pokreće *pipeline*. Z koordinata tražene točke dobije se pomoću funkcije *get_distance()* kojoj su ulazni argumenti cjelobrojne vrijednosti koordinata X i Y. Funkcija *get_point_cloud()* vrati funkciji *get_z()* traženu vrijednost te se *pipeline* prekida.

# 7. ROBOTSKI OPERATIVNI SUSTAV

Zajednica robotičara napravila je impresivan napredak zadnjih godina. Pouzdan i ne preskup robotski hardver, od mobilnih robota koji se kreću po tlu, preko raznih dronova pa sve do humanoidnih robota, sve je mnogo pristupačnije nego ikada prije. Što je još impresivnije, zajednica robotičara razvila je algoritme koji pomažu robotima dostići određeni stupanj autonomnosti. Unatoč tome, roboti i dalje predstavljaju veliki izazov za razvojne programere. Kako bi se pokušali smanjiti ti izazovi, napravljen je robotski operativni sustav – ROS. ROS je operativni sustav otvorenog koda za robote. On osigurava sve što se očekuje od jednog operativnog sustava, uključujući podršku za hardver, upravljanje uređajima na niskoj razini, implementaciju najčešće korištenih funkcionalnosti, prosljeđivanje poruka među procesima i drugo. Također, sadrži alate i biblioteke za pribavljanje, izgradnju, pisanje i pokretanje koda preko raznih računala. [25]

## 7.1. Implementacija algoritma u ROS-u

Radi korištenja kreiranog algoritma za detekciju i praćenje RONNA-e na mobilnom robotu, potrebno je implementirati ga u robotski operativni sustav. Čvor u robotskom operativnom sustavu označava izvršni dio koji je povezan na ROS mrežu. Kako bi se algoritam implementirao u ROS potrebno je kreirati dvije python skripte, *Publisher* i *Subscriber*, koje predstavljaju čvorove. *Publisher* čvor objavljuje podatke, a *Subscriber* se pretplaćuje na njih i dohvaća ih.

### *7.1.1. Kreiranje Publisher čvora*

Svaki ROS čvor pisan u Pythonu ima na početku ovu deklaraciju. Ta linija osigurava da se skripta pokreće kao Python skripta.

```
1 #!/usr/bin/env python
```

Zatim je potrebno učitati određene biblioteke koje su potrebne za pokretanje skripte. Za pisanje ROS čvora potrebna je biblioteka *rospy*, a za objavljivanje tekstualnog zapisa (stringa) potrebna je biblioteka *std_msgs.msg*.

```
3 import rospy
4 from std_msgs.msg import String
```

Sljedeći dio koda definira sučelje čvora koji šalje podatke prema ostatku ROS-a. Sljedeća linija deklarira da čvor objavljuje podatke na temu koristeći tekstualni tip poruke (*string*).

```
7 pub = rospy.Publisher('chatter', String, queue_size=10)
```

Sljedeća linija je jako bitna jer sprema u *rospy* ime čvora. Sve dok rospy nema tu informaciju, ne može pokrenuti komunikaciju s ROS *Masterom*. Odabiranje *anonymus* argumenta osigurava da čvor ima jedinstveno ime dodavanjem nasumično odabranog broja na kraj imena.

```
8 rospy.init_node('talker', anonymous=True)
```

Sljedeća linija kreira objekt *rate* iz klase *Rate*. To osigurava da se petlja vrti željenim tempom. Budući da je argument 10 to znači da će se petlja ponoviti 10 puta u sekundi.

```
9 rate = rospy.Rate(10) # 10hz
```

Sljedeće linije su standardni dio *rospy* konstrukcije. Nakon provjere je li *rospy* ugašen započinje petlja. Potrebno je provjeriti je li ugašen da se zna treba li program prestati s radom. Zatim se definira varijabla koja se ispisuje u terminalu te objavljuje na temi. Petlja poziva metodu *rate.sleep()* koja osigurava da se petlja vrti točno onoliko puta u sekundi koliko treba.

```
 10 while not rospy.is_shutdown():
 11     hello_str = "hello world %s" % rospy.get_time()
 12     rospy.loginfo(hello_str)
 13     pub.publish(hello_str)
 14     rate.sleep()
```

Sljedeći korak je kreiranje čvora za primanje poruka. [26]

### *7.1.2. Kreiranje Subscriber čvora*

Kod za čvor za primanje poruka sličan je kodu čvora za slanje poruka. Razlika je u novom povratnom mehanizmu za pretplaćivanje na poruke.

Sljedeće linije osiguravaju da se čvor pretplati na odgovarajuću temu. Kada je poruka primljena, povratni mehanizam je pozvan s porukom kao prvim argumentom. [26]

```
 15 rospy.init_node('listener', anonymous=True)
 16
 17 rospy.Subscriber("chatter", String, callback exiting until this node
is stopped
 18 rospy.spin()
```

### *7.1.3. Testiranje izrađenih čvorova*

Prije povezivanja algoritma za detekciju objekata s ROS čvorovima, testirat će se napisani kod s porukom *hello world* i trenutnim vremenom. Kod pokretanja ROS čvorova, prvi korak je pokrenuti *roscore*.

```
roscore
```

Nakon toga, u novom terminal prozoru pokreće se *publisher* čvor koji objavljuje podatke. Prvo je potrebno pozvati *bash* datoteku, a zatim se pomoću naredbe *rosrun* poziva čvor.

```
source ./devel/setup.bash
rosrun beginner_tutorials talker.py
```

Sada je čvor *Publisher* pozvan i u terminalu se ispisuju poruke koje on šalje. Sljedeće treba pokrenuti čvot *Subscriber*.

```
rosrun beginner_tutorials listener.py
```

### *7.1.4. Povezivanje algoritma s ROS čvorovima*

Izrađene čvorove potrebno je prilagoditi algoritmu za detekciju i prepoznavanje objekta. tema se naziva *pubRonnaCord*, a umjesto stringa koji se prosljeđuje u drugi čvor, definira se točka (Point) koja ima zapis od tri koordinate. Naziv čvora je *pubCord*.

```
pub = rospy.Publisher('pubRonnaCord', Point, queue_size=10)
    newCord = Point()
    rospy.init_node('pubCord', anonymous=True)
    rate = rospy.Rate(10) # 10hz
```

U beskonačnoj petlji definira se varijabla *tempCord* u koju se zapisuje izlaz iz glavne funkcije modula *detect_ronna.py*. Varijabla prima objekt koji se sastoji od tri koordinate.

```
while not rospy.is_shutdown():

        tempCord = startProgram()

        if tempCord:
            newCord.x = tempCord[0]
            newCord.y = tempCord[1]
            newCord.z = tempCord[2]
        else:
            newCord.x = 0
            newCord.y = 0
            newCord.z = 0

        pub.publish(newCord)
        rate.sleep()
```

Slijedi provjera jesu li koordinate dobivene te ako jesu, zapisuju se u novu varijablu *newCord*. Ako koordinate nisu dobivene sprema se vrijednost nula za svaku jer u suprotnom bi se spremila vrijednost *None* pa bi se čvor koji čita srušio.

### *7.1.5. Pokretanje i testiranje*

Da bi se algoritam pokrenuo unutar ROS-a potrebno je otvoriti tri terminal prozora. U prvom je potrebno pozvati *roscore* naredbu, odnosno glavni čvor koji osigurava komunikaciju između drugih čvorova.

```
roscore
```

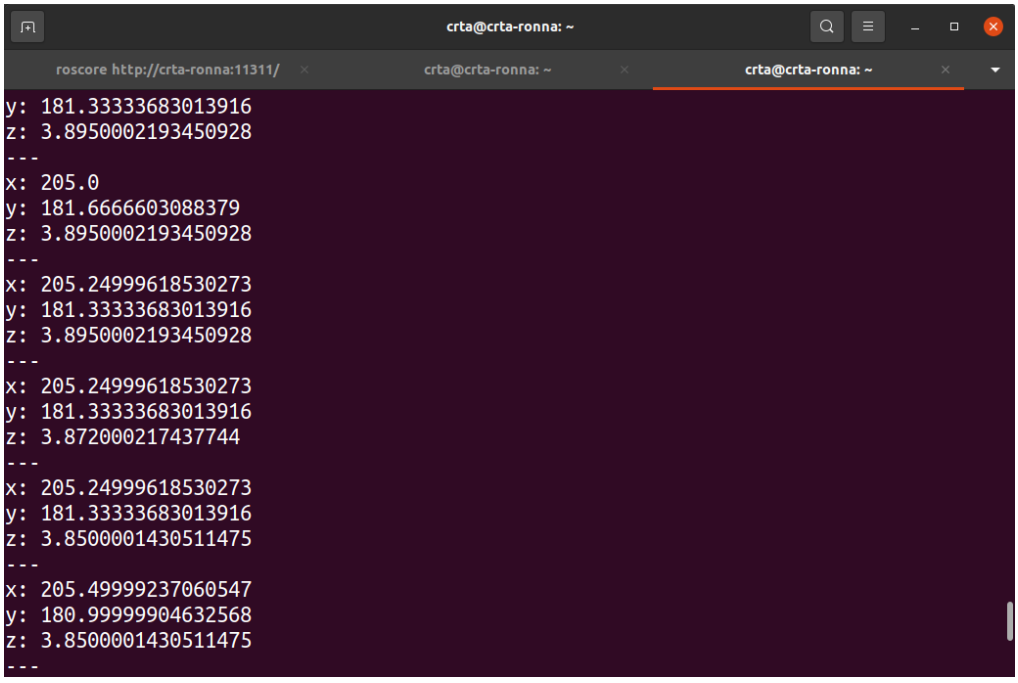
U drugom terminal prozoru pokreće se čvor koji šalje poruke, odnosno *Publisher*. Prije pokretanja tog čvora obavezno se mora pozvati *bash* datoteka koja sadrži osnovne postavke.

```
source catkin_ws/devel/setup.bash
rosrun beginner_tutorials pubRonnaCord.py
```

Konačno, u trećem terminal prozoru poziva se čvor za čitanje poruka. U tom prozoru se ispisuju koordinate detektiranog objekta.

```
rostopic echo /pubRonnaCord
```

Na slici 33 je prikazano kao izgleda prikaz koordinata detektiranog objekta u terminal prozoru.




**Slika 33. Prikaz ispisivanja koordinata pomoću ROS-a**

# 8. VALIDACIJA ALGORITMA ZA DETEKCIJU I PRAĆENJE OBJEKTA

## 8.1. Validacija algoritma za detekciju i praćenje

Nakon što je algoritam izrađen, potrebno ga je validirati u laboratoriju. Da bi se algoritam za detekciju i praćenje validirao potrebno je ponovno složiti postav s kolicima, prijenosnim računalom i s kamerom. Slika 34 prikazuje navedeni postav za validaciju u laboratoriju i medicinski robot RONNA koji je objekt detekcije i praćenja. Cilj validacije je ispitati kako radi algoritam, jesu li podatci koje daje kao rezultate točni, koji su mu nedostatci te koji su mogući uzroci tih nedostataka.

**Slika 34. Postav za validaciju algoritma za detekciju i praćenje**

Kada se pokrene algoritam za detekciju i praćenje, na ekranu računala kontinuirano se pojavljuju rezultati detekcije objekta. Ti rezultati obuhvaćaju koordinate težišta objekta i to X i Y koordinatu u pikselima te Z koordinatu kao udaljenost težišta objekta od kamere u metrima. Ispitivanje je provedeno tako da je neuronavigacijski sustav RONNA stajao na jednom mjestu, a postav se gurao prema njemu i oko njega. Prilikom ispitivanja nastojalo se obuhvatiti sve strane RONNA-e iz različitih kutova. Na slici 35 prikazani su rezultati validacije algoritma.

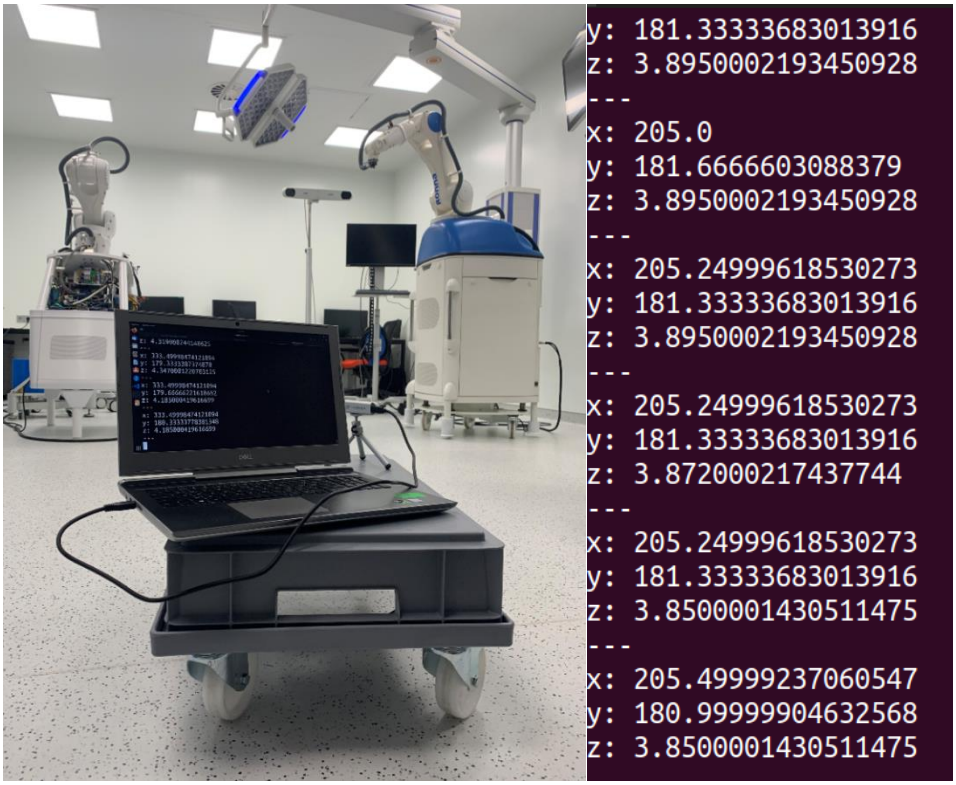
```
Fusing layers...
Model Summary: 224 layers, 7053910 parameters, 0 gradients
image 1/1 C:\Users\Malnar\Desktop\PythonScripts\Photos\2021_07_04_20_45_27.jpg: X: 248 Y: 69 Z: 6.297000408172607
YOLOv5  2021-4-26 torch 1.9.0+cpu CPU

Fusing layers...
Model Summary: 224 layers, 7053910 parameters, 0 gradients
image 1/1 C:\Users\Malnar\Desktop\PythonScripts\Photos\2021_07_04_20_45_32.jpg: X: 236 Y: 71 Z: 5.615000247955322
YOLOv5  2021-4-26 torch 1.9.0+cpu CPU

Fusing layers...
Model Summary: 224 layers, 7053910 parameters, 0 gradients
image 1/1 C:\Users\Malnar\Desktop\PythonScripts\Photos\2021_07_04_20_45_38.jpg: X: 236 Y: 71 Z: 5.264000415802002
YOLOv5  2021-4-26 torch 1.9.0+cpu CPU

Fusing layers...
Model Summary: 224 layers, 7053910 parameters, 0 gradients
image 1/1 C:\Users\Malnar\Desktop\PythonScripts\Photos\2021_07_04_20_45_44.jpg: X: 236 Y: 71 Z: 5.434000492095947
YOLOv5  2021-4-26 torch 1.9.0+cpu CPU

Fusing layers...
Model Summary: 224 layers, 7053910 parameters, 0 gradients
image 1/1 C:\Users\Malnar\Desktop\PythonScripts\Photos\2021_07_04_20_45_49.jpg: X: 236 Y: 71 Z: 5.478000164031982
YOLOv5  2021-4-26 torch 1.9.0+cpu CPU

Fusing layers...
Model Summary: 224 layers, 7053910 parameters, 0 gradients
image 1/1 C:\Users\Malnar\Desktop\PythonScripts\Photos\2021_07_04_20_45_55.jpg: X: 236 Y: 71 Z: 5.434000492095947
YOLOv5  2021-4-26 torch 1.9.0+cpu CPU

Fusing layers...
Model Summary: 224 layers, 7053910 parameters, 0 gradients
image 1/1 C:\Users\Malnar\Desktop\PythonScripts\Photos\2021_07_04_20_46_00.jpg: X: 236 Y: 71 Z: 5.910000324249268
YOLOv5  2021-4-26 torch 1.9.0+cpu CPU

Fusing layers...
Model Summary: 224 layers, 7053910 parameters, 0 gradients
image 1/1 C:\Users\Malnar\Desktop\PythonScripts\Photos\2021_07_04_20_46_05.jpg: X: 236 Y: 71 Z: 5.759000301361084
YOLOv5  2021-4-26 torch 1.9.0+cpu CPU

Fusing layers...
Model Summary: 224 layers, 7053910 parameters, 0 gradients
image 1/1 C:\Users\Malnar\Desktop\PythonScripts\Photos\2021_07_04_20_46_10.jpg: X: 236 Y: 71 Z: 5.963000297546387
YOLOv5  2021-4-26 torch 1.9.0+cpu CPU
```

**Slika 35. Rezultati validacije algoritma za detekciju i praćenje objekta**

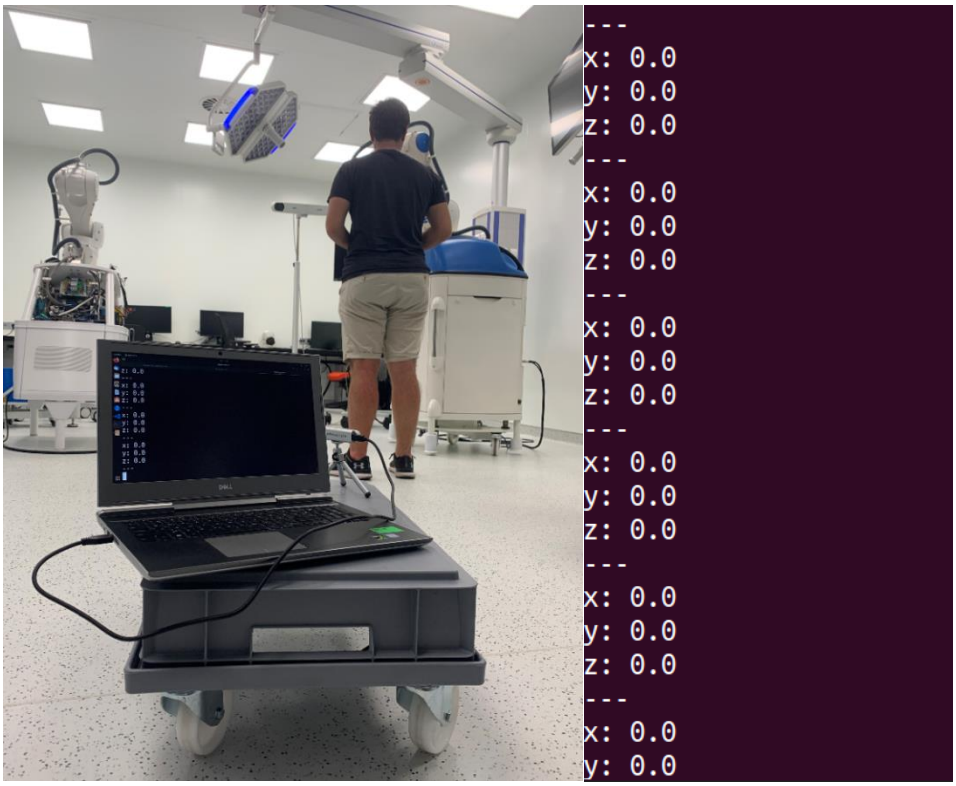
Iz dobivenih rezultata vidljivo je da je neuronska mreža YOLO v5 dobro istrenirana i da se objekt prepoznaje u svakom prolazu kroz beskonačnu petlju. Međutim, kada se pogleda iznos prostorne koordinate vidljivo je da njeno određivanje nije skroz precizno. Dok postav miruje, X i Y koordinata su stabilne, ali vrijednost Z koordinate varira. Osim toga, dok se postav pomiče prema robotu, očekivano bi bilo da vrijednost Z koordinate opada, ali to se ne događa nego ona i dalje varira između manjih i većih vrijednosti.

## 8.2. Validacija algoritma u ROS okruženju

Kada je algoritam za detekciju objekta integriran u ROS, potrebno ga je validirati u laboratoriju. Postupak validacije bio je isti kao u prethodno opisanom slučaj bez ROS-a. Postav se sastojao od prijenosnog računala s Intel RealSense kamerom na postolju s kotačima. Na slici 36 prikazana je detekcija objekta kada ništa nije na putu između objekta i kamere, a na slici 37 vidi se da se objekt ne detektira ako se između nalazi prepreka.




**Slika 36. Prikaz rezultata detekcije kada se robot vidi bez prepreka**




**Slika 37. Prikaz rezultata detekcije s preprekom**

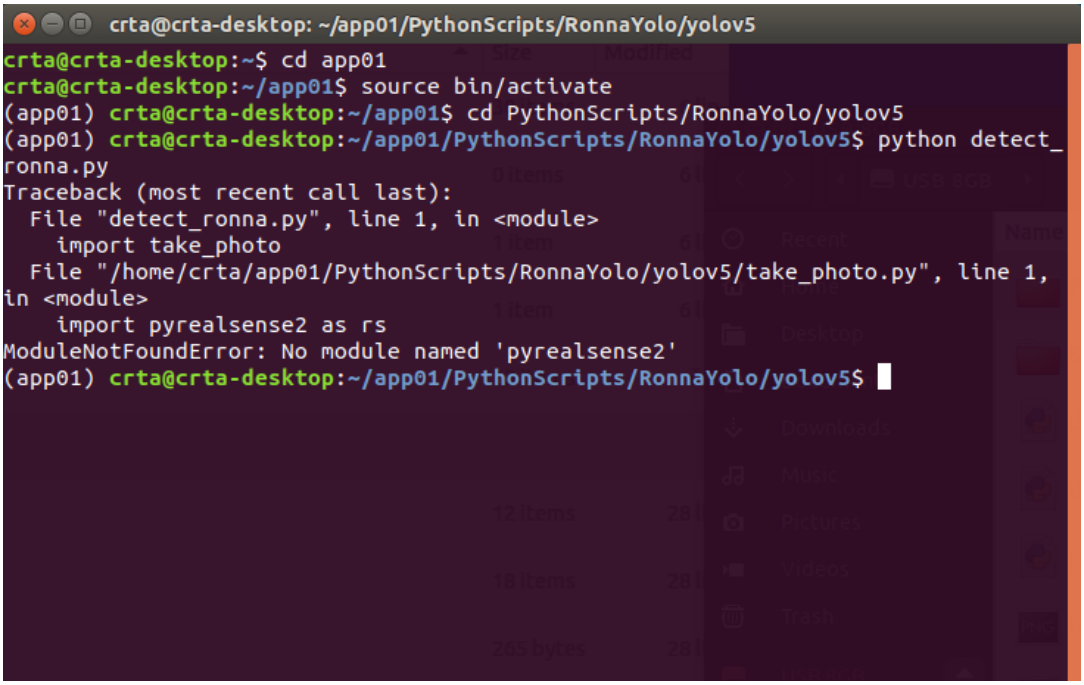
## 8.3. Komentari rezultata validacije

Prilikom validacije algoritma u laboratoriju utvrđeno je da algoritam u uvjetima kada nema prepreka između kamere i objekta koji je potrebno detektirati u većini slučajeva ispisuje koordinate. Simulirana je situacija kada je osoba stala između kamere i objekta pa su se na zaslonu računala umjesto koordinata ispisivale nule. Ipak, bilo je slučajeva kada nije bilo prepreke između kamere i objekta, a ipak su se ispisivale nule umjesto koordinata. Osim toga, u nekim slučajevima iznos Z koordinate bio je približno točan stvarnoj vrijednosti, dok je u nekim slučajevima značajno odstupao.

## 8.4. Mogućnosti za poboljšanje algoritma

Kao što je prikazano u radu, napravljen je algoritam za detekciju i praćenje željenog objekta koji kao izlaz daje koordinate udaljenost težišta objekta u prostoru od kamere u metrima te X i Y koordinatu u pikselima. Prilikom validacije algoritma uočeno je da dubinska koordinata koju algoritam daje nije stabilna i potpuno točna. U nekim slučajevima dok kamera miruje, vrijednost Z koordinate varira. Pretpostavka je da se to događa zbog toga što YOLO v5 model ne postavi granični okvir svaki put na isto mjesto, a Z koordinata direktno je ovisna o težišti središtu okvira. Kao jedno od poboljšanja ovog algoritma svakako bi bilo dodatno istraživanje oblaka točaka i detektiranje problema koji se trenutno javlja s određivanjem dubine točke. Jedno od mogućih rješenja je da se ne uzima samo jedna točka za određivanje dubine nego skup točaka oko nje.

Druga stvar koja bi se trebala omogućiti je pokretanje algoritma na Nvidia Jetson Xavier NX platformi. U sklopu ovog zadatka to se nije izvelo jer algoritam zahtijeva *pyrealsense2* paket koji se na Xavier NX platformu nije mogao jednostavno instalirati. Budući da za taj paket i platformu Xavier NX s arch64 arhitekturom računala nije moguća instalacija putem pip ili sudo funkcije. Prilikom ručne instalacije više puta dolazilo je do grešaka. Budući da modul *take_photo.py* zahtijeva taj paket za slikanje s kamerom, jetson NX nije se koristio za validaciju algoritma. Slika 38 prikazuje grešku zbog koje nije moguće pokrenuti algoritam na Xavier NX platformi.




**Slika 38. Prikaz greške o izostanku modula *pyrealsense2***

Iz tog razloga, validacija algoritma provedena je na prijenosnom računalu na kojem je bio instaliran operativni sustav Linux Ubuntu.

# 9. ZAKLJUČAK

U današnje vrijeme algoritmi za detekciju i praćenje objekata doživjeli su značajan razvoj i postaju sve više zastupljeni u svakodnevnom životu. Autonomna vožnja postala je sadašnjost i sve više proizvođača okreće se tome, a upravo su ti algoritmi za autonomnu vožnju jedni od najkompliciranijih. Osim za autonomnu vožnji, algoritmi za detekciju koriste se i u drugim područjima poput mobilne robotike. YOLO mreža, od kada se pojavila u zadnjih par godina, značajno je pojednostavnila izradu algoritama za detekciju objekata u realnom vremenu i rasprostranila njihovu primjenu.

U ovom radu prikazan je postupak treniranja YOLO v5 mreže slikama medicinskog robota RONNA, dohvaćanje prostora točaka i određivanje dubine na slici pomoću Intel RealSense kamere, izrada algoritma za detekciju i praćenje objekta, njegova implementacija u ROS-u te validacija u laboratoriju. Posao dovoženja i odvoženja medicinskog robota u operacijsku salu i iz nje na prvi pogled možda zvuči trivijalno, ali kada se uzme u obzir koliko takvih radnji ljudi obavljaju svaki dan, pritom zanemarujući jedni druge, pokazuje se kao dobro pogođen problem koji treba riješiti. Već sada strojevi umjesto ljudi peru odjeću i posuđe, usisavaju kuću, sjeckaju povrće i slično, a alati poput YOLO v5 postavili su temelje da se ta pomoć strojeva ljudima podigne na jednu višu razinu i uđe u sferu onoga što je do jučer bilo nezamislivo.

# LITERATURA

[1] Siciliano, Khatib, „Handbook of Robotics“, Springer, 2016.

[2] Siegwart, Nourbakhsh, Scaramuzza, „Introduction to Autonomous Mobile Robots“, The MIT Press, 2004.

[3] http://roboticsandautomationnews.com/2019/10/29/gideon-brothers-raises-e2-65-million-in-new-funding-and-partners-with-db-schenker/26507/, pristupljeno 4. 7. 2021.

[4] Jerbić et al., „RONNA G4-Robotic Neuronavigation: A Novel Robotic Navigation Device for Stereotactic Neurosurgery“, Handbook of Robotics and Image-Guided Surgery, 2019.

[5] https://www.croatiaweek.com/croatia-develops-successful-robotised-neurosurgery-system/, pristupljeno 2. 7. 2021.

[6] https://www.windowscentral.com/kinect-windows-v2-sensor-sales-end-developers-can-use-xbox-one-version, pristupljeno 2. 7. 2021.

[7] Intel RealSense D400 Series Product Family, Datasheet, Revision 005, 2019.

[8] https://www.fabtolab.com/intel-realsense-depth-cam-d435, pristupljeno 2. 7. 2021.

[9] https://www.loc.gov/preservation/digital/formats/fdd/fdd000501.shtml, pristupljeno 7. 7. 2021.

[10] Redmon, Divvala, Girshick, Farhadi, „You only Look Once: Unified, Real-Time Object Detection“, IEEE Xplore, 2016.

[11] Bolf, Jerbić I., „Primjena umjetnih neuronskih mreža“, Kem. Ind., 2006.

[12] Čupić, „Umjetne neuronske mreže“, FER, 2001.

[13] Yegnanarayana B., „Artificial Neural Networks“, PHI, 2006.

[14] https://www.presenceorb.com/2020/01/mvsense-objectreconition/, pristupljeno 4. 7. 2021.

[15] Fang, Guo, Chen, Zhou, Ye, „Accurate and Automated Detection of Surface Knots on Sawn Timbers Using YOLO-V5 Model“, BioResources, 2021.

[16] https://blog.roboflow.com/labelimg/, pristupljeno 8. 4. 2021.

[17] https://www.tutorialspoint.com/google_colab/google_colab_quick_guide.htm, pristupljeno 3. 7. 2021.

[18] https://www.antratek.com/nvidia-jetson-xavier-nx-developer-kit, pristupljeno 4. 7. 2021.

[19] https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-nx/, pristupljeno 2. 7. 2021.

[20] https://blog.generationrobots.com/en/comparison-of-nvidia-jetson-nanocomputers/, pristupljeno 4. 7. 2021.

[21] https://developer.nvidia.com/embedded/jetson-modules, pristupljeno 5. 7. 2021.

[22] https://developer.nvidia.com/embedded/learn/get-started-jetson-xavier-nx-devkit - write, pristupljeno 14. 5. 2021.

[23] https://uoa-eresearch.github.io/eresearch-cookbook/recipe/2014/11/26/python-virtual-env/, pristupljeno 30. 5. 2021.

[24] https://linuxize.com/post/how-to-install-python-3-8-on-ubuntu-18-04/, pristupljeno 30. 5. 2021.

[25] Jason M. O'Kane, „A Gentle Introduction to ROS“, Independently published, 2013.

[26] http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29, pristupljeno 6. 7. 2021.

# PRILOZI

I. Skripta detect_ronna.py

II. Skripta take_photo.py

III. Skripta detect_yolo.py

IV. Skripta pointcloud.py

## detect_ronna.py

```
import os, sys
currentdir = os.path.dirname(os.path.realpath(__file__))
parentdir = os.path.dirname(currentdir)
sys.path.append(parentdir)

import detect_ronna.take_photo
import detect_ronna.detect_yolo
import os
import shutil


def startProgram():
    while True:
        #putanja do foldera u koji se spremaju slike za detekciju
        path = '/home/crta/catkin_ws/src/beginner_tutorials/src/de-
tect_ronna/ronna_photos'
        #putanja do .pt datoteke s težinama za detekciju
        weights = "/home/crta/catkin_ws/src/beginner_tutorials/src/de-
tect_ronna/weights/box_best_130.pt"


        if os.path.exists(path): #provjera postoji li već folder za
spremanje slika
            shutil.rmtree(path)  #ako postoji, folder se briše

        os.makedirs(path) #kreiranje foldera za spremanje slika

        detect_ronna.take_photo.main(path) #pozivanje funkcije za okidanje
slika
        if os.listdir(path): #provjera nalazi li se nešto u folderu za
slike
            line_final = detect_ronna.detect_yolo.detect(path, weights)
#ako se nalazi onda se pokreće detekcija
            return line_final
            #print(line_final)
        else:
            return[0,0,0]
        #return line_final

if __name__ == '__main__':
    startProgram()
```

## take_photo.py

```
import pyrealsense2 as rs
import numpy as np
import cv2
import time
import os


def main(path):
    pipeline = rs.pipeline()
    config = rs.config()

    config.enable_stream(rs.stream.color, 1280, 720, rs.format.bgr8, 30)

    profile = pipeline.start(config)

    align_to = rs.stream.color
    align = rs.align(align_to)

    frames = pipeline.wait_for_frames()

    aligned_frames = align.process(frames)

    color_frame = aligned_frames.get_color_frame()

    color_image = np.asanyarray(color_frame.get_data())

    # definiranje naziva slike
    image_name = str(time.strftime("%Y_%m_%d_%H_%M_%S")) +  '.jpg'

    # spremanje slike
    cv2.imwrite(os.path.join(path, image_name), color_image)

    pipeline.stop()

if __name__ == '__main__':
    main()
```

## detect_yolo.py

```
import argparse
import time
from pathlib import Path

import os, sys
currentdir = os.path.dirname(os.path.realpath(__file__))
parentdir = os.path.dirname(currentdir)
sys.path.append(parentdir)

import detect_ronna.pointcloud

import cv2
import torch
import torch.backends.cudnn as cudnn
from numpy import random
import pyrealsense2 as rs

from detect_ronna.models.experimental import attempt_load
from utils.datasets import LoadStreams, LoadImages
from utils.general import check_img_size, check_requirements, check_imshow,
non_max_suppression, apply_classifier, \
    scale_coords, xyxy2xywh, strip_optimizer, set_logging, increment_path,
save_one_box
from utils.plots import plot_one_box
from utils.torch_utils import select_device, load_classifier,
time_synchronized


def detect(source, weights):
    view_img = False
    save_txt = True
    img_size = 640
    project = 'runs/detect'
    name = 'exp'
    imgsz =source, weights, view_img, save_txt, img_size
    nosave = False
    exist_ok = False
    save_img = not nosave and not source.endswith('.txt')  # save inference
images
    webcam = source.isnumeric() or source.endswith('.txt') or
source.lower().startswith(
        ('rtsp://', 'rtmp://', 'http://', 'https://'))

    # Directories
    save_dir = increment_path(Path(project) / name, exist_ok=exist_ok)  #
increment run
    (save_dir / 'labels' if save_txt else save_dir).mkdir(parents=True,
exist_ok=True)  # make dir

    # Initialize
    device = ''
    set_logging()
```

```
    device = select_device(device)
    half = device.type != 'cpu'  # half precision only supported on CUDA

    # Load model
    model = attempt_load(weights, map_location=device)  # load FP32 model
    stride = int(model.stride.max())  # model stride
    imgsz = check_img_size(img_size, s=stride)  # check img_size
    if half:
        model.half()  # to FP16

    # Second-stage classifier
    classify = False
    if classify:
        modelc = load_classifier(name='resnet101', n=2)  # initialize
        modelc.load_state_dict(torch.load('weights/resnet101.pt',
map_location=device)['model']).to(device).eval()

    # Set Dataloader
    vid_path, vid_writer = None, None
    if webcam:
        view_img = check_imshow()
        cudnn.benchmark = True  # set True to speed up constant image size
inference
        dataset = LoadStreams(source, img_size=imgsz, stride=stride)
    else:
        dataset = LoadImages(source, img_size=imgsz, stride=stride)

    # Get names and colors
    names = model.module.names if hasattr(model, 'module') else model.names
    colors = [[random.randint(0, 255) for _ in range(3)] for _ in names]

    # Run inference
    if device.type != 'cpu':
        model(torch.zeros(1, 3, imgsz,
imgsz).to(device).type_as(next(model.parameters())))  # run once
    t0 = time.time()
    for path, img, im0s, vid_cap in dataset:
        img = torch.from_numpy(img).to(device)
        img = img.half() if half else img.float()  # uint8 to fp16/32
        img /= 255.0  # 0 - 255 to 0.0 - 1.0
        if img.ndimension() == 3:
            img = img.unsqueeze(0)

        # Inference
        t1 = time_synchronized()
        augment = False
        pred = model(img, augment=augment)[0]

        # Apply NMS
        pred = non_max_suppression(pred, conf_thres=0.4, iou_thres=0.45,
agnostic=False)
        t2 = time_synchronized()

        # Apply Classifier
        if classify:
            pred = apply_classifier(pred, modelc, img, im0s)
```

```
        # Process detections
        for i, det in enumerate(pred):  # detections per image
            if webcam:  # batch_size >= 1
                p, s, im0, frame = path[i], '%g: ' % i, im0s[i].copy(),
dataset.count
            else:
                p, s, im0, frame = path, '', im0s.copy(), getattr(dataset,
'frame', 0)

            p = Path(p)  # to Path
            save_path = str(save_dir / p.name)  # img.jpg
            txt_path = str(save_dir / 'labels' / p.stem) + ('' if
dataset.mode == 'image' else f'_{frame}')  # img.txt
            s += '%gx%g ' % img.shape[2:]  # print string
            gn = torch.tensor(im0.shape)[[1, 0, 1, 0]]  # normalization
gain whwh
            if len(det):
                # Rescale boxes from img_size to im0 size
                det[:, :4] = scale_coords(img.shape[2:], det[:, :4],
im0.shape).round()

                # Print results
                for c in det[:, -1].unique():
                    n = (det[:, -1] == c).sum()  # detections per class
                    s += f"{n} {names[int(c)]}{'s' * (n > 1)}, "  # add to
string

                save_conf = True
                # Write results
                for *xyxy, conf, cls in reversed(det):
                    if save_txt:  # Write to file
                        xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) /
gn).view(-1).tolist()  # normalized xywh
                        line = (cls, *xywh, conf) if save_conf else (cls,
*xywh)  # label format
                        with open(txt_path + '.txt', 'a') as f:
                            f.write(('%g ' * len(line)).rstrip() % line +
'\n')
                            coords = xywh[:2] #spremanje podataka o x i y
koordinati u novu varijablu
                            line_final = rosReturnType(coords) #pozivanje
funkcije get_z s varijablom koja sadrži x i y koordinate
                            return line_final

                    save_crop = False
                    if save_img or save_crop or view_img:  # Add bbox to
image
                        hide_labels = False
                        hide_conf = False
                        c = int(cls)  # integer class
                        label = None if hide_labels else (names[c] if
hide_conf else f'{names[c]} {conf:.2f}')

                        plot_one_box(xyxy, im0, label=label,
color=colors[c], line_thickness=3)
```

```
                        if save_crop:
                            save_one_box(xyxy, im0s, file=save_dir /
'crops' / names[c] / f'{p.stem}.jpg', BGR=True)

            # Stream results
            if view_img:
                cv2.imshow(str(p), im0)
                cv2.waitKey(1)  # 1 millisecond

            # Save results (image with detections)
            if save_img:
                if dataset.mode == 'image':
                    cv2.imwrite(save_path, im0)
                else:  # 'video' or 'stream'
                    if vid_path != save_path:  # new video
                        vid_path = save_path
                        if isinstance(vid_writer, cv2.VideoWriter):
                            vid_writer.release()  # release previous video
writer
                        if vid_cap:  # video
                            fps = vid_cap.get(cv2.CAP_PROP_FPS)
                            w = int(vid_cap.get(cv2.CAP_PROP_FRAME_WIDTH))
                            h = int(vid_cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
                        else:  # stream
                            fps, w, h = 30, im0.shape[1], im0.shape[0]
                            save_path += '.mp4'
                        vid_writer = cv2.VideoWriter(save_path,
cv2.VideoWriter_fourcc(*'mp4v'), fps, (w, h))
                    vid_writer.write(im0)

    if save_txt or save_img:
        s = f"\n{len(list(save_dir.glob('labels/*.txt')))} labels saved to
{save_dir / 'labels'}" if save_txt else ''


# dodana funkcija za dobivanje z koordinate
def get_z(coord):
    # jedinične koordinate preračunavaju se u koordinate kamere u pikselima
    X = coord[0]*640
    Y = coord[1]*480

    # provjera je li otkriven objekt
    if X:
        # pozivanje funkcije za određivanje dubine piksela
        Z = detect_ronna.pointcloud.get_point_cloud(X, Y)
        # ispisivanje sve tri koordinate težišta objekta
    return "\nRonna detected! \n\nX: {}\nY: {}\nZ: {}".format(int(X),
int(Y), Z)

# dodana funkcija za dobivanje z koordinate
def rosReturnType(coord):
    # jedinične koordinate preračunavaju se u koordinate kamere u pikselima
    X = coord[0]*640
    Y = coord[1]*480
    # provjera je li otkriven objekt
    if X:
```

```
        # pozivanje funkcije za određivanje dubine piksela
        Z = detect_ronna.pointcloud.get_point_cloud(X, Y)
        # ispisivanje sve tri koordinate težišta objekta
        tempReturn = [X, Y, Z]
        return tempReturn
    else:
        return [1,2,3]
if __name__ == '__main__':
    detect()
```

## point_cloud.py

```
import pyrealsense2 as rs


def get_point_cloud(x, y):
    pipeline = rs.pipeline()
    config = rs.config()
    pipeline.start(config)
    profile = pipeline.get_active_profile()

    frames = pipeline.wait_for_frames()

    depth_frame = frames.get_depth_frame()

    # pozivanje funkcije za određivanje dubine
    zDepth = depth_frame.get_distance(int(x), int(y))

    # slanje vrijednosti u skriptu detect_yolo.py
    return(zDepth)

    pipeline.stop()


if __name__ == '__main__':
    get_point_cloud()
```

## pubRonnaCord.py

```
#!/usr/bin/env python3

import rospy
import sys

import os, sys
currentdir = os.path.dirname(os.path.realpath(__file__))
parentdir = os.path.dirname(currentdir)
sys.path.append(parentdir)

from std_msgs.msg import String
from geometry_msgs.msg import Point
from src.detect_ronna.detect_main import startProgram

def talker():
    pub = rospy.Publisher('pubRonnaCord', Point, queue_size=10)
    newCord = Point()
    rospy.init_node('pubCord', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():

        tempCord = startProgram()

        if tempCord:
            newCord.x = tempCord[0]
            newCord.y = tempCord[1]
            newCord.z = tempCord[2]
        else:
            newCord.x = 0
            newCord.y = 0
            newCord.z = 0

        pub.publish(newCord)
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```