Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Martin Turk

Navigacija poljskega robota

diplomsko delo

interdisciplinarni univerzitetni
študijski program prve stopnje
računalništvo in matematika

Mentor: izr. prof. dr. Danijel Skočaj

Ljubljana, 2018

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Robotika postaja vse pomembnejša tehnologija tudi v kmetijstvu. V zadnjih letih je opazen trend povečanja razvoja in uporabe poljskih robotov, ki na kmetijskih površinah opravljajo najrazličnejša opravila. Avtonomnost in inteligentnost takšnih robotov je zelo različna. Najnaprednejši so sposobni samostojne navigacije med poljščinami in samostojnega opravljanja zahtevanih nalog. V diplomski nalogi nadgradite mobilnega robota, da bo imel podobne sposobnosti. Mobilni robot naj bo sposoben samostojne navigacije med vrstami koruze s pomočjo procesiranja podatkov pridobljenih s senzorjem LiDAR in kamero.

Iskreno se zahvaljujem mentorju izr. prof. dr. Danijelu Skočaju za vso pomoč in hitro odzivnost. Zahvalil bi se tudi ekipi Tafr, ki me je sprejela in mi omogočila izdelavo diplomskega dela. Izjemno hvaležen sem svoji družini, ki me je podpirala in pomagala med izdelavo diplomske naloge.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
AMCL	adaptive Monte Carlo localization	adaptivna lokalizacija Monte Carlo
BLDC	brushless direct current motor	brezščetinski motor z enosmernim tokom
CNC	computer numerical control	računalniško računsko upravljanje
CNN	convolutional neural network	konvolucijska nevronska mreža
DBSCAN	density based spatial clustering of applications with noise	prostorsko gručenje podatkov s šumom na osnovi gostote
FRE	field robot event	tekmovanje s poljskimi roboti
GPS	global positioning system	globalni sistem pozicioniranja
HDMI	high definition multimedia interface	visokoločljiv multimedijski vmesnik
HMI	human machine interface	vmesnik med človekom in računalnikom
IMU	inertial measurement unit	inertna merska enota
LCD	liquid cristal display	tekočekristalni zaslon
LIDAR	light detection and ranging	svetlobna detekcija in merjenje razdalje
LIPO	lithium polymer battery	litij polimer baterije
OSRF	open source robotics foundation	odprtokodna robotska fundacija

PID	proportional integral derivative controller	regulator PID
RADAR	radio detection and ranging	radijska detekcija in merjenje razdalje
RAM	random access memory	bralno pisalni pomnilnik
ROS	robot operating system	robotski operacijski sistem
SAIL	Stanford artificial intelligence laboratory	laboratorij za umetno inteligenco na Stanfordu
SLAM	simultaneous localization and mapping	simultana lokalizacija in mapiranje
SSD	solid state drive	pomnilniški disk
TAFR	telematic autonomous field robot	telematski avtonomni poljski robot
URDF	unified robot description format	zedinjen format za opis robota
VESC	opensource electronic speed controller	odprtokodni električni regulator hitrosti
XML	extensible markup language	razširljivi označevalni jezik

Povzetek

Naslov: Navigacija poljskega robota

Avtor: Martin Turk

Robotski sistemi so danes že tako napredni, da so ponekod zamenjali ljudi pri opravljanju dela in predstavljamo si lahko, da bo tega v prihodnosti še več. Da so roboti inteligentni in lahko bolje opravljajo delo kot človek, se morajo biti sposobni sami navigirati v okolju in se odločati v različnih situacijah. Razvili smo inteligentnega robota za tekmovanje FRE *(ang. Field Robot Event)*, ki se avtonomno navigira po polju koruze. V tem diplomskem delu smo se osredotočili na razvoj aplikacije za avtonomno vožnjo robota s pomočjo senzorja LIDAR ter prikazali, kako bi vožnjo izboljšali s pomočjo zaznavanja koruze s kamero. Podatki pridobljeni s senzorjem LIDAR, imajo običajno veliko šuma, zato jih moramo obdelati z algoritmom za gručenje DBSCAN. V zaključku opišemo eksperimentalne rezultate ter naštejemo nekaj možnih izboljšav in načrtov za prihodnost.

Ključne besede: avtonomna navigacija, robot, LIDAR, DBSCAN, robotski operacijski sistem, konvolucijske nevronske mreže.

Abstract

Title: Navigation of field robot

Author: Martin Turk

Advanced robot systems have already substituted the human factor in many fields of work and we can only imagine what the future will bring. In order for a robot to be intelligent, to perform better at executing tasks than human, they need to be autonomous. We developed a robot for Field Robot Event competition, where it needs to be able to autonomously navigate through a corn field. In this thesis, we developed a robot application which is responsible for autonomous navigation with LIDAR sensor and proposed how to improve the navigation with camera recognition. The data collected with LIDAR usually contains a lot of noise, which we reduce with DBSCAN clustering algorithm. In conclusion, we present the results from the competition, DBSCAN algorithm, and camera recognition. We also propose a few improvements and future plans.

Keywords: autonomous navigation, robot, LIDAR, DBSCAN, Robot Operation System, CNN.

Poglavje 1

Uvod

1.1 Motivacija

Zaradi neučinkovitosti in nenatančnosti ljudi ter močnega napredka v tehnologiji, se je na nekaterih področjih v našem življenju močno povečala prisotnost robotskih sistemov. Ker želimo delo najbolj optimizirati in so ljudje utrudljivi in zmotljivi, so robotski sistemi v nekaterih vrstah dela zamenjali ljudi. Za primer lahko vzamemo proizvodnjo ali skladišča, ki so lahko v velikem delu avtomatizirana, tako da lahko delujejo brez večjih posegov človeka v sistem. Po drugi strani pa za nekatere naloge ljudje nismo dovolj natančni (npr. v zdravstvu) ali pa so okoliščine dela za nas preveč nevarne (npr. deaktivacija bomb, posredovanje v naravnih katastrofah, kot so požari in poplave itd.).

Uporaba inteligentnejših robotskih sistemov je koristna tudi z ekonomskega in ekološkega vidika. Dobri primeri so uporaba pesticidov, vode in gnojila v kmetijstvu. Ogromne količine porabe vode in pesticidov je možno optimizirati z uporabo robotov, kar je posledično ceneje in bolj ekološko. S pomočjo strojnega učenja, umetne inteligence in računalniškega vida se lahko robot nauči prepoznavati bolezen na rastlinah ali primankljaje vode in hranilnih snovi. Z zbranimi informacijami lahko nato na primer ustrezno zalivamo polje po potrebi – ponekod več, drugje manj, prilagodimo uporabo

pesticidov ali označimo bolne rastline za odstranitev. Z aplikacijo robotike v kmetijstvu se ukvarja veliko ekip in podjetij. Eno izmed takih podjetij je Rowbot, ki je razvilo manjšega avtonomnega robota, ki se lahko vozi med ozkimi vrstami visoke koruze. Podjetje ponuja rešitve za kontrolirano dovajanje hranilnih snovi koruzi kjer je potrebno in sejanje rastja za kolobarjenje ob optimalnem času. Za večja polja roboti delujejo tudi v skupinah.

Večina robotskih sistemov ni avtonomnih, saj so vnaprej programirani za izvajanje točno določene funkcije. O avtonomnosti lahko govorimo, ko je robot sposoben opravljati naloge sam brez dodatne pomoči človeka dalj časa, ko zna sam reagirati na situacije v okolju in njim ustrezno ukrepati.

Avtonomen robot se mora torej samostojno navigirati v danem okolju. To je možno doseči s pomočjo senzorjev, kot so GPS *(ang. Global Positioning System)*, RADAR *(ang. RAdio Detection And Ranging)*, LIDAR *(ang. Light Detection And Ranging)*, IMU *(ang. Intertial Measurement Unit)* in kamere ter odometrija.

Ker senzorji sistemu posredujejo velike količine podatkov, jih mora znati interpretirati tako, da si z njimi lahko pomaga. Dober primer je LIDAR, ki vrača podatke v obliki zaznanih točk iz okolice z veliko šuma. Z različnimi metodami lahko nato podatke uredimo in preoblikujemo.

Obstaja več tekmovanj, kjer ekipe tekmujejo s svojimi avtonomnimi roboti. Eno takih tekmovanj, ki smo se ga z ekipo udeležili letos v Nemčiji, je FRE *(ang. Field Robot Event)*, kjer mora biti robot sposoben navigirati po polju in opravljati razne naloge. Tekmovalna robota ekipe Tafr sta TAFR 1.0 na sliki 1.2 in TAFR 2.0 na sliki 1.1. Problem pri navigiranju robota po polju koruze je ozka vozna površina in razvejanost listov, ki lahko hitro zmede senzorje, robot pa lahko posledično zapelje v vrsto koruze in jo poškoduje. Ker je robot širok približno toliko kot pot, ves čas zaznava ovire, ki so zelo blizu, zato se mora gibati čim bolj gladko in brez naglih in hitrih zavojev. Za to skrbi regulator PID *(ang. Proportional–Integral–Derivative controller)*, ki s pomočjo izračunanih napak med vožnjo prilagaja hitrosti koles.

Razvejanost listov koruze lahko povzroči velik problem pri štetju vrst

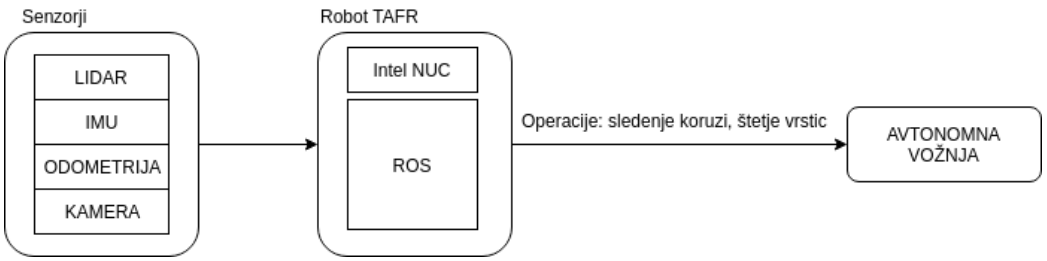
koruze. Če zapeljemo iz poti med vrstama koruze in želimo zapeljati v katerokoli drugo, moramo vrste koruze zaznati in šteti. V primeru ko imata koruzi sosednjih vrst močno razvejane liste, lahko senzor LIDAR na sredini poti to zazna kot novo vrsto. Problem je možno rešiti s prepoznavo koruze s pomočjo kamere.

Slika 1.1: Robot TAFR 2.0.

Slika 1.2: Robot TAFR 1.0.

1.2 Definicija problema

Problem, ki smo si ga zastavili za diplomsko delo, je implementacija navigacije robota na polju koruze za tekmovanje FRE. Pri vožnji med vrstami se robot navigira večinoma s senzorjem LIDAR, pri štetju vrst pa uporablja še IMU in odometrijo. Posebej je implementirano še zaznavanje vrst koruze s kamero, ki še ni del sistema, saj robot še nima vgrajene grafične kartice. Implementirati je bilo potrebno vozlišča za interpretacijo podatkov in operacije, ki omogočajo avtonomnost robota. Zaradi razvejanosti koruze imamo v podatkih, ki jih vrne senzor LIDAR, šum, zato jih moramo urediti z algoritmom za gručenje.



Slika 1.3: Oris rešitve problema.

1.3 Zgradba diplomske naloge

Diplomska naloga je sestavljena iz 6 poglavij. V uvodnem napovemo temo in definicijo problema ter predstavimo grob oris rešitve. Drugo poglavje opiše robota, ki ga je letos razvila ekipa Tafr ter predstavi senzorje, ki smo jih uporabili. V tretjem poglavju teoretično opišemo algoritem za gručenje, ki odpravi šum iz podatkov, konvolucijske nevronske mreže, s pomočjo katerih naučimo model za prepoznavanje koruze in implementacijo modela za zaznavanje. Četrto poglavje se osredotoči na implementacijo aplikacije TAFR in operacije, ki so zadolžene za avtonomno navigacijo robota. V predzadnjem poglavju predstavimo rezultate s tekmovanja, algoritma za gručenje in modela za prepoznavanje koruze iz slike. V zaključku diplomske naloge predstavimo še naslednje cilje in možnosti izboljšav.

Poglavje 2

Mobilni robot

Ekipa Tafr, študentje Fakultete za elektrotehniko in Fakultete za računalništvo in informatiko Univerze v Ljubljani so razvili že svojega drugega poljskega robota TAFR 2.0 *(ang. Telematic Autonomous Field Robot)*. Robot je agilen, zmožen dinamične vožnje po relativno zahtevnem neravnem terenu. Idealna velikost omogoča lažje manevriranje po polju, priključke raznih sistemov in senzorjev, ki jih uporablja za avtonomno vožnjo in opravljanje nalog; hkrati za delovanje ne potrebuje veliko moči. Za dober izgled robota je poskrbel oblikovalec iz ekipe, ki je narisal več dizajnov robota, preden smo pršli do želene oblike. Končen model je viden na sliki 2.1.

Slika 2.1: Končni računalniški model robota TAFR 2.0 brez zaščitne maske.

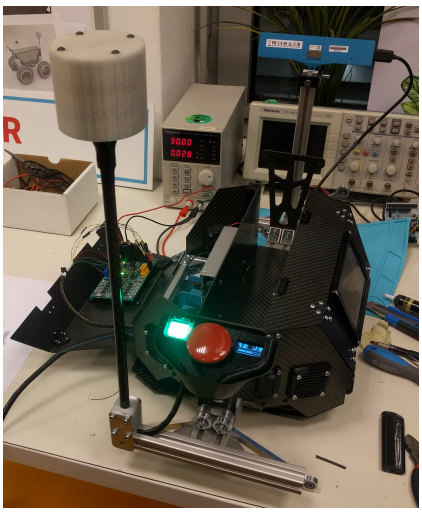
2.1 Mehanika

Robota poganjajo štirje zmogljivi motorji, ki so hkrati tudi terenska kolesa z grobim profilom, ki omogočajo vožnjo tudi po bolj zahtevni podlagi. Kolesa so preko vzmetenja (slika 2.2) pritrjena na močno podvozje, ki je bilo ročno sestavljeno iz aluminijastih in nekaj karbonskih delov. Vzmetenje je pomembno za istočasen in enakomeren oprijem vseh koles na neravnih tleh, kar je ključnega pomena, ker robot zavija tako, da se ena stran koles vrti hitreje kot druga. Veliko aluminijastih delov je bilo ročno načrtovanih in izrezkanih na CNC *(ang. Computer Numerical Control)* stroju. Na podvozje je pritrjeno ohišje, sestavljeno iz več karbonskih plošč, ki so med seboj pritrjene z vijaki. Za odpiranje stranic ohišja smo uporabili plastične zatiče, ki so tiskani s 3D tiskalnikom. Zaradi uporabe lahkih materialov robot tehta le 35 kg. Na desni zgornji stranici ohišja je pritrjen večji LCD zaslon na dotik, ki je namenjen prikazu diagnostike robota in izvajanje ukazov. Spodnji in zgornji del ohišja, kot je vidno na sliki 2.3, je možno odpreti na levi in desni strani, kar omogoča hiter dostop do vseh komponent. V sredini ohišja je hitro dostopna odprtina za baterijo, kar nam omogoča hitro menjavo. Na zadnji strani ohišja je pritrjena manjša kontrolna plošča z gumboma za vklop in majhnim LCD zaslonom, ki prikazuje podatke o napajanju in stanju zagona robota. Na podvozje sta pritrjeni še luč in palica za priklop senzorjev.

2.2 Elektronika

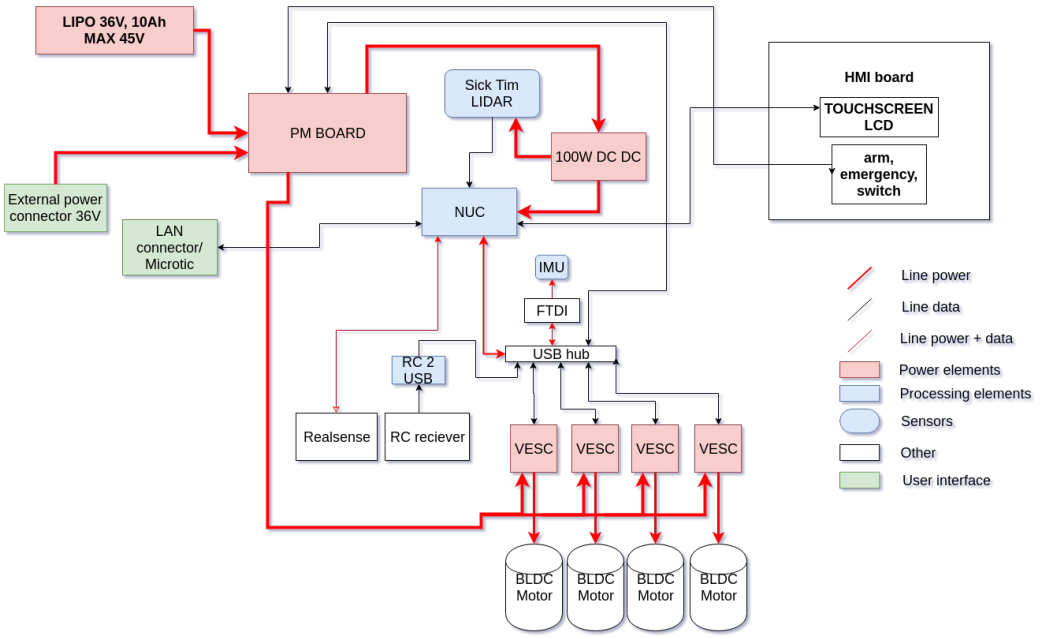
Delovanje elektronike in povezava med električnimi elementi, senzorji in računalnikom je prikazana na sliki 2.4. Robota napaja 36V LIPO *(ang. Lithium Polymer Battery)* baterija s kapaciteto 10Ah, kar navadno zadostuje za 2 do 3 ure delovanja robota. Večino moči porabijo štirje 500W BLDC *(ang. Brushless Direct Current)* motorji, ki so kontrolirani s pomočjo odprtokodnih VESC gonilnikov. Robot ima torej 2kW moči in doseže najvišjo hitrost pri 35 km/h, zato je bilo potrebno moč motorjev za avtonomno vožnjo omejiti. Vsak motor ima tudi magnetni enkoder. Ti lahko obrat kolesa izmerijo do desetinke

Slika 2.2: Vzmetenje robota pomaga pri lažjem premagovanju ovir in boljšemu merjenju podatkov.



Slika 2.3: Ohišje robota za elektroniko, sestavljeno iz karbonskih plošč.

stopinje natančno. Poleg odometrije enkoderji skrbijo tudi za konstantno hitrost robota ne glede na breme (npr. če se vozimo po hribu navzgor, enkoderji motorjem dovajajo več moči za ohranjanje hitrosti). PM board *(ang. Power Management)* je elektronsko vezje, načrtovano po meri, ki skrbi za porazdelitev električne napetosti med komponentami.



Slika 2.4: Elektronska shema robota TAFR 2.0.

HMI *(ang. Human-Machine Interface)* board je elektronsko vezje, ki skrbi za prikaz diagnostike, informacij o omrežju, podatkov o bateriji, stanje robota (vožnja, pavza ali ustavljen) na zaslonu. Nadzoruje tudi vklop robota. Zaradi varnosti ima robot stikalo za prekinitev moči iz baterije robotu, stikalo za vklop in izklop motorjev, ki ima tudi vlogo stikala v sili in varnostno stikalo za ponoven vklop motorjev (poskrbi, da ob vklopu motorjev robot ne pobegne).

2.3 Senzorji ter ostali gradniki robota

Vsi senzorji, ki so našteti v nadaljevanju, so povezani s centralnim računalnikom. Kamera je nameščena na aluminijast profil, ki je namenjen priključkom. Senzor LIDAR je z zaščito postavljen nižje med sprednji dve kolesi.

Intel NUC i7 je glavni računalnik in igra vlogo možganov robota z Linux Kubuntu operacijskim sistemom. Ima integrirano grafično kartico in dodatno vsebuje še 16 GB RAM pomnilnik in 64 GB SSD trdi disk.

Prikazan je na sliki 2.5.

Slika 2.5: Intel NUC i7.

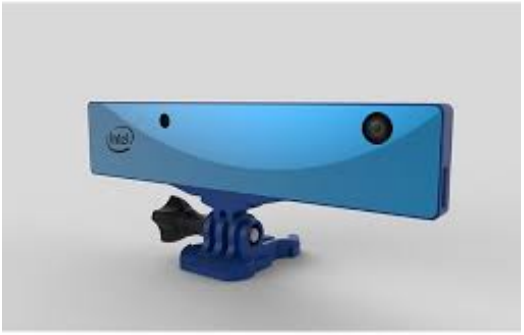
Intel RealSense Zr300 je globinska stereo kamera (2.6), ki ima integrirano ribje oko *(ang. fisheye)* in IMU. Slike lahko zajema z ločljivostjo 1920 x 1080 pri 30 fps (okvirjih na sekundo), 640 x 480 pri 15, 30 ali 60 fps in 320 x 240 pri 30 ali 60 fps.

2D LiDAR SICK TiM551 je senzor iz slike 2.7, ki meri razdaljo do objektov v prostoru s pomočjo infrardečih svetlobnih žarkov. Skenira lahko pod kotom 270° do 10 metrov daleč s frekvenco 15 Hz.

IMU (Inertial Measurement Unit) je naprava, sestavljena iz giroskopa, pospeškometra in magnetometra, ki meri kotno hitrost in pospešek. Iz meritev lahko izračunamo kot nagiba, zavoja in obrata robota.

Odometrija Odometrijo računajo magnetni enkoderji podjetja RLS.

HDMI LCD 800×480 LCD zaslon na dotik, ki prikazuje diagnostiko robota in omogoča izvajanje ukazov.



Slika 2.6: Intel RealSense Zr300.

Slika 2.7: 2D LIDAR SICK TiM551.

Poglavje 3

Teoretična podlaga

Za lažje razumevanje razdelka o implementaciji aplikacije TAFR (4.2) potrebujemo nekaj teoretičnega predznanja. V tem poglavju bomo opisali kako deluje algoritem za gručenje DBSCAN ter predstavili kaj so konvolucijske nevronske mreže, ki jih uporabimo za učenje modela za prepoznavo koruze s slik.

3.1 Gručenje 3D točk

DBSCAN je algoritem za gručenje, ki so ga leta 1996 predstavili Martin Ester, Hans-Peter Kriegel, Jörg Sander in Xiaowei Xu [8]. Pogosto se ga uporablja v strojnem učenju in podatkovnem rudarjenju.

V splošnem algoritem združuje točke, ki so si blizu v prostoru – združuje regije točk glede na gostoto. Kot parametre algoritmu podamo:

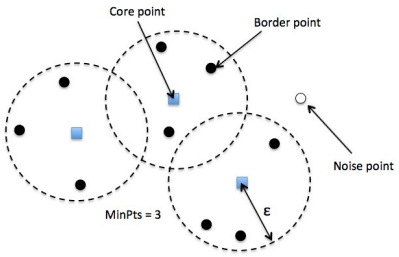
- množico točk, na kateri se gručenje izvaja,
- epsilon (ε), ki predstavlja velikost radia v katerem iščemo sosednje točke. Lahko si predstavljamo, da je vsaka točka center krožnice z radijem ε,
- minimalno število točk znotraj krožnice, da še tvorimo gručo (`minTck`).

Vsako točko nato algoritem klasificira v enega od treh razredov:

1. **Glavna točka:** Ima najmanj minTck točk znotraj svoje krožnice.

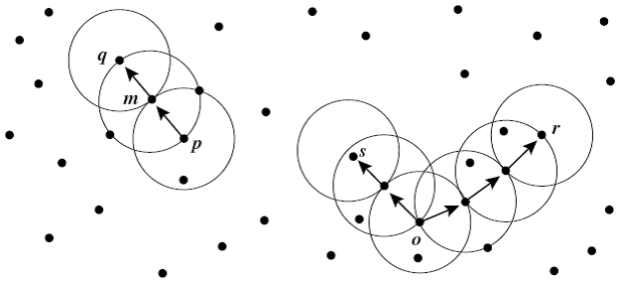
2. **Robna točka:** je v radiju glavne točke, vendar ima manj kot minTck točk v svoji krožnici.

3. **Šumna točka:** je točka, ki ni niti glavna niti robna.



Slika 3.1: Primer klasifikacije z minTck = 3 [6].

Vsaka glavna točka sestavlja eno gručo skupaj z robnimi točkami. Pravimo da sta dve točki neposredno povezani preko gostote, če je ena od točk glavna, druga pa znotraj krožnice. Glavne točke se lahko tako preko gostote verižijo v gručo.



Slika 3.2: Primer veriženja glavnih točk. Na levi strani vidimo, da je p povezana neposredno preko gostote z m in m z q. Veriga glavnih točk p $\rightarrow$ m $\rightarrow$ q tvori eno gručo. Podobno sta v gručo povezani s in r. [6]

3.2 Konvolucijske nevronske mreže

Ljudje neprestano zbiramo informacije iz okolja in jih podzavestno procesiramo. Med gledanjem avtomatično zaznamo objekte v vidnem polju glede na pretekle izkušnje, jih spremljamo in se ustrezno odločamo [2]. Medtem ko zaznavanje človeku ne predstavlja izziva, je za računalnik izjemno zahtevna. Računalnik se mora tako kot človek naučiti prepoznavati predmete tako, da algoritmu podamo veliko učnih slik. To znamo učinkovito rešiti z uporabo konvolucijskih nevronskih mrež, ki so vrsta globokih nevronskih mrež.

Slik, zaradi kompleksnosti tipa podatkov, ne moremo učinkovito klasificirati z uporabo umetnih nevronskih mrež, medtem ko CNN to rešijo tako, da sliko razdelijo na več vidnih polj, ki se prekrivajo in za vsakega poiščejo značilke. Z vsakim višjim nivojem operiramo z vidnim poljem, ki zajema večji del vhodne slike, kar pomeni da lahko iščemo značilke za bolj kompleksne vzorce.

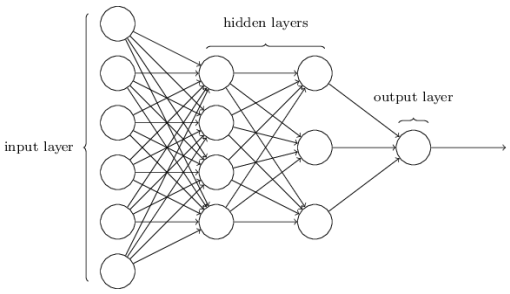
Kot smo že omenili, so vhodni podatki slike predstavljeni v treh dimenzijah $W \times H \times C$, kjer sta širina W in višina H prostorski dimenziji, C pa predstavlja število barvnih kanalov.

V tem podpoglavju bomo opisali umetne nevronske mreže, zgradbo in glavne nivoje konvolucijske nevronske mreže ter zgradbo implementiranega modela za zaznavanje koruze.

3.2.1 Umetne nevronske mreže

Umetna nevronska mreža je poln, usmerjen graf brez ciklov, ki ima n nivojev: vhodni in izhodni nivo ter *n-2* skritih nivojev. Vsak skriti nivo ima poljubno število nevronov, medtem ko ima vhodni nivo toliko nevronov, kot imajo vhodni podatki atributov, izhodni nivo pa toliko, kot je klasifikacijskih razredov. Vrednost vsakega nevrona v izhodnem nivoju si lahko predstavljamo kot verjetnost klasifikacije vhoda v ta razred. Primer večnivojske umetne nevronske mreže lahko vidimo na sliki 3.3.

Vsak nevron ima aktivacijsko vrednost, ki jo izračunamo tako, da seštejemo



Slika 3.3: Umetna nevronska mreža z vhodnim in izhodnim ter dvema skritima nivojema. [3]

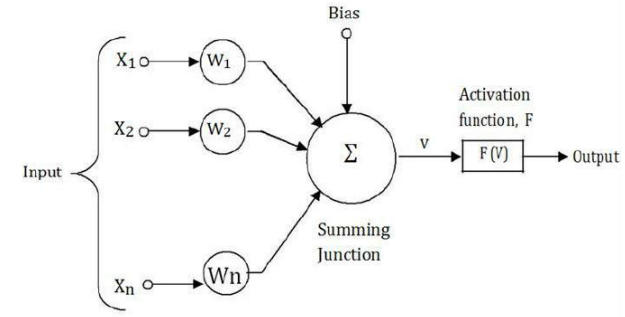
utežene vrednosti vseh vhodov in rezultat nato normaliziramo z vnaprej določeno aktivacijsko funkcijo. Aktivacijske funkcije so pomemben del nevronskih mrež, saj vpeljejo faktor nelinearnosti. Brez nelinearnosti nevronska mreža ne bi mogla aproksimirati nelinearnih funkcij, ki ločujejo razrede podatkov. Najpogosteje uporabljene aktivacijske funkcije so sigmoida (3.1), hiperbolični tangens (3.2) in funkcija ReLU (3.3).

$$S(x) = \frac{1}{1 + e^{-x}} \tag{3.1}$$

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3.2}$$

$$f(x) = max(0, x) \tag{3.3}$$

Pri učenju umetnih nevronskih mrež igra pomembno vlogo tudi odmik *(ang. bias value)*, ki jo nevron doda že uteženi vsoti. Vrednost odmika dovoljuje, da je izhod nevronske mreže 0, čeprav je vhodna vrednost blizu 1. Dodajanje vrednosti odmika omogoča premikanje aktivacijske funkcije po osi x. Elementi nevrona in postopek interpretiranja vhoda v izhod je prikazan na sliki 3.4 in z enačbo (3.4), kjer je X_i izhod i-tega nevrona, F aktivacijska funkcija, b_i odmik nevrona i in $W_{j,i}$ utež med nevronoma j in i.



Slika 3.4: Elementi nevrona v umetni nevronski mreži. [4]

$$X_{out} = F(\sum_{j} X_j W_{j,out} + b_{out}) \tag{3.4}$$

3.2.2 Vzvratno razširjanje napake

Z vzvratnim razširjanjem prilagodimo izračunane uteži nevronske mreže, tako da je izhod bližje želenemu. Temu procesu pravimo učenje. V več iteracijah učenja zmanjšujemo globalno napako nevronske mreže, kar pomeni, da v resnici iščemo množico uteži, pri katerih bo imela umetna nevronska mreža najmanjšo globalno napako na učni množici.

Z vzvratnim razširjanjem napake računamo gradiente, ki jih potem pri gradientnem spustu uporabimo za minimizacijo kriterijske funkcije.

Gradientni spust

Gradientni spust je eden najpogosteje uporabljenih iterativnih optimizacijskih algoritmov v strojnem učenju. Je metoda, ki poišče lokalni ali globalni minimum funkcije napake oziroma napako pri učenju minimizira. Pogosto uporabljena funkcija napake je vsota kvadratov (3.5), ki nam pove, kako blizu pravilnim vrednostim so napovedane.

$$C(x) = \sum_j \frac{1}{2}(x_j - r_j)^2, \tag{3.5}$$

kjer je x_j napovedana vrednost, r_j dejanska vrednost in j število izhodov nevronske mreže [7].

Gradient je vektor parcialnih odvodov funkcije, ki kaže v smeri največjega vzpona funkcije, zato se moramo pri gradientnem spustu premikati v nasprotni smeri gradienta in ustrezno popravljati uteži nevronske mreže. To storimo tako, da za vsak nevron izračunamo parcialni odvod funkcije napake $C(x)$ glede na vsako utež $W_{j,i}$ in jo posodobimo po formuli 3.6.

$$W_{j,i} = W_{j,i} - \alpha \frac{\partial C}{\partial W_{j,i}}, \tag{3.6}$$

kjer je α stopnja učenja, ki nam pomaga preprečevati divergenco (v primeru strmih spustov ni nujno, da najdemo globalni minimum [1]). α nam torej pove, da moramo spreminjati uteži sorazmerno z gradientom.

3.2.3 Konvolucija

Konvolucija je matematična operacija nad funkcijama vidnega polja f in filtra g, katere rezultat je nova funkcija, definirana v diskretnem prostoru kot:

$$(f * g)(x) = \sum_{u=-\infty}^{\infty} g(u)h(x-u) \tag{3.7}$$

Konvolucijo lahko posplošimo tudi na več dimenzij in je ključnega pomena pri globokem učenju, saj lahko z različnimi filtri sliko konvuliramo in iz nje izluščimo značilke.

3.2.4 Nivoji

Medtem ko si lahko umetno nevronsko mrežo predstavljamo kot polno povezan usmerjen graf, je arhitektura CNN sestavljena iz več različnih nivojev, ki skrbijo za obdelavo kompleksnejšega tipa vhodnih podatkov in klasifikacijo. Učenje CNN je manj zahtevno, kot učenje umetnih nevronskih mrež, saj imajo manj nevronov in več skritih nivojev [14]. V tem podpoglavju bomo našteli in opisali glavne nivoje konvolucijskih nevronskih mrež, izhod katerih je nato povezan z umetno nevronsko mrežo (3.2.1), ki sliko klasificira.

Konvolucijski nivo

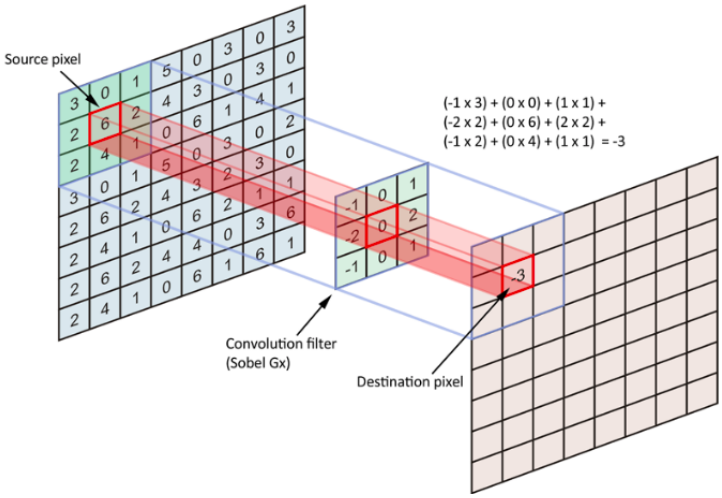
Konvolucijski nivo je najpomembnejši gradnik CNN. Konvolucijo izvedemo med filtrom in vhodnimi podatki; rezultat je matrika značilk. Delu vhoda, ki ga v nekem koraku prekriva filter, pravimo dovzetno polje *(ang. receptive field)*. Konvolucija poteka tako, da filter premikamo po vhodu in v vsakem koraku zmnožimo matriki v vidnem polju po komponentah ter jih seštejemo. Rezultat v vsakem koraku zapišemo na ustrezno mesto v matriko značilk. Običajno je konvolucija v CNN implementirana kot korelacija (slika 3.5).

Primer konvolucije je podan v dveh dimenzijah. Tipično se izvaja v 3D. Filtri se raztezajo čez celotno globino izhoda prejšnjega nivoja. Rezultat konvolucije je torej matrika značilk. Ker na vsakem konvolucijskem nivoju uporabimo več različnih filtrov imamo kot rezultat toliko matrik značilk, kolikor je filtrov. To pomeni, da je globina izhoda konvolucijskega nivoja enaka številu filtrov. Filtre pri CNN si lahko predstavljamo kot uteži pri umetnih nevronskih mrežah. Predstavljajo del mreže, ki se uči in so na začetku naključno inicializirani.

Konvolucijskemu nivoju podamo naslednje hiperparametre:

1. velikost filtra, ki je tipično matrika velikosti 3×3, 5×5 ali 7×7,

2. velikost koraka *(ang. stride)* pove, za koliko se bo v vsaki iteraciji premaknil filter na vhodu,



Slika 3.5: Implementacija konvuliranja vhoda s filtrom pri običajnih CNN. Filter se pomika po vhodu in izračunane vrednosti zapisuje v nov nivo. [5]

3. zapolnjevanje z ničlami *(ang. padding)* definira število dodanih ničelnih robov. Z velikostjo koraka se matrika značilk manjša, kar pomeni, da izhod konvolucijskega nivoja ne bo imel enake prostorske dimenzije kot vhod. Za ohranjanje iste prostorske dimenzije vhodni matriki dodamo ustrezno število ničelnih robov.

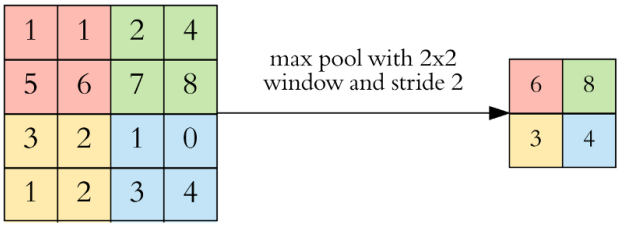
Aktivacijski nivo

Tako kot pri umetnih nevronskih mrežah moramo tudi pri CNN vpeljati faktor nelinearnosti. Izhod konvolucijskega nivoja običajno normaliziramo tako, da ga pošljemo skozi ReLU aktivacijsko funkcijo (3.3). Aktivacijski nivo velikosti podatkov ne spremeni [13].

Nivo združevanja

Izhod konvolucijskega nivoja je običajno velikih dimenzij. S postopkom združevanja *(ang. pooling)* zmanjšamo število parametrov, kar zmanjša čas učenja in odpravlja prilagajanje učnim primerom *(ang. overfitting)*.

Najpogosteje uporabljen postopek združevanja je združevanje maksimalnih vrednosti *(ang. max pooling)*, ki je prikazan na sliki 3.6. Postopek je tak, da z vnaprej določeno velikostjo okna korakamo po vhodu in ohranjamo le največje vrednosti, ostale pa zanemarimo. Ohranjene vrednosti združimo v nov vhod, ki ga sprejme naslednji konvolucijski nivo. S postopkom konvolucije in združevanja najdemo le najboljše značilke. Tako smo zmanjšali količino podatkov potrebnih za obdelavo.



Slika 3.6: Združevanje maksimalnih vrednosti z velikostjo koraka 2 in velikostjo okna 2×2. [5]

Nivo izpuščanja

Zaradi velikega števila parametrov se lahko mreža, še posebej v polno povezanem nivoju, med učenjem preveč prilagodi učnim primerom. To pomeni, da se med učenjem v mreži aktivirajo le določeni nevroni, ostali pa so prezrti. Problem lahko rešimo s slojem izpuščanja, ki v vsakem koraku učenja deaktivira naključno število nevronov. To izboljša generalizacijo *(ang. generalization)*, saj prisili, da se nivo z različnimi nevroni uči istih značilk.

Nivo sploščitve

V zadnjem koraku predprocesiranja podatkov pred polno povezanim nivojem nastopi nivo sploščitve, ki izhod prejšnjih nivojev v obliki matrik spremeni v enodimenzionalni vektor.

Polno povezan nivo

Po predprocesiranju podatkov s konvolucijskimi nivoji in združevanjem se arhitektura običajnih CNN konča s polno povezanim nivojem, ki ni nič drugega kot umetna nevronska mreža, opisana v razdelku 3.2.1.

3.3 Implementacija modela za zaznavanje

Za implementacijo modela smo uporabili verzijo višje nivojske knjižnice za globoko učenje *Keras*, ki deluje na ogrodju za strojno učenje *Tensorflow*. Model je naučen na slikah dveh klasifikacijskih razredov in sicer koruze ter poti.

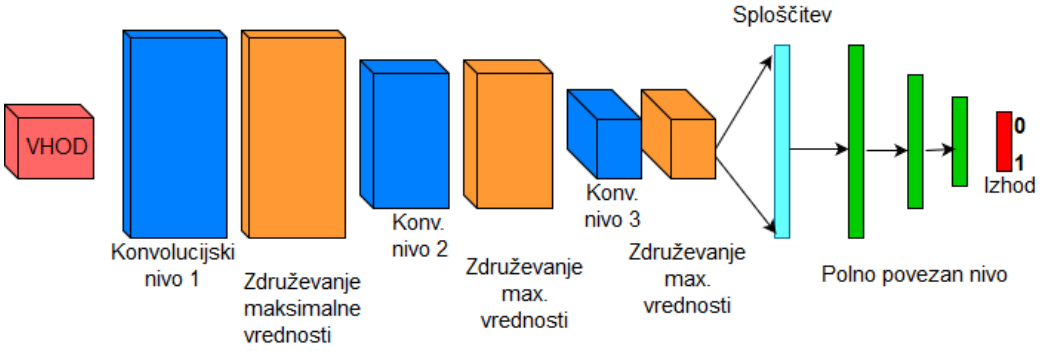
3.3.1 Priprava podatkov

Slike za učenje in evalvacijo smo pridobili iz posnetkov, iz katerih smo ročno izrezali primere koruze in poti. Slikam smo nato spremenili velikost na 50×50 slikovnih elementov. Nato smo slike koruz označili kot razred 0 in jih razdelili v dve množici – množico učnih primerov in množico za evalvacijo. Podobno smo storili s slikami poti, le da smo jih označili kot razred 1.

3.3.2 Arhitektura modela

Arhitektura konvolucijske nevronske mreže (slika 3.7), na kateri smo model učili, je sestavljena iz nivojev, opisanih v podpoglavju 3.2. Na začetku imamo tri konvolucijske nivoje, kjer vsakemu sledi nivo maksimalnega združevanja vrednosti. Prvi konvolucijski nivo ima 50 filtrov velikosti 7×7, drugi 25 filtrov velikosti 5×5 in tretji 10 filtrov velikosti 3×3. Med vsakim konvolucijskim nivojem in nivojem združevanja je še nivo aktivacije, kjer vrednosti normaliziramo z aktivacijsko funkcijo ReLU.

Sledita nivo izpuščanja, za preprečitev prilagoditve učenja na učne primere in nivo sploščitve, kjer podatke uredimo v eno dimenzionalni vektor, ki ga podamo polno povezanemu nivoju.



Slika 3.7: Arhitektura nivojev modela.

Klasifikacijski del mreže sestavljajo štirje polno povezani nivoji. V prvem imamo 150 nevronov in za aktivacijo uporabljamo funkcijo ReLU. Drugi nivo je sestavljen iz 90 nevronov in prav tako uporablja aktivacijsko funkcijo ReLU. Predzadnji nivo ima 50 nevronov, zadnji pa 2: toliko, kot je razredov. Oba za aktivacijsko funkcijo uporabljata sigmoido.

Poglavje 4

Implementacija

Programiranje robotov ni enostavno. Nadzorovati je potrebno veliko senzorjev, skrbeti za komunikacijo med vsemi komponentami, izvajati zahtevnejše računske operacije in algoritme, paziti na varnost in še veliko več. Vse naloge je potrebno izvajati istočasno in v realnem času, kar nam delo še oteži.

Zato je bilo razvito ogrodje ROS, ki skrbi za izvajanje kode, tako da se lahko osredotočimo na pomembnejše naloge. Osnov delovanja ROSa se bomo dotaknili v nadaljevanju, implementacija aplikacije pa bo predstavljena v razdelku 4.2.

Za avtonomno vožnjo robota je zadolžen skupek programov, algoritmov, konfiguracijskih datotek, orodij in knjižnic. Programiran je večinoma s programskima jezikoma *C++* in *Python*.

4.1 Robotski Operacijski Sistem (*ROS*)

ROS je ogrodje oziroma platforma za izdelovanje robotskih aplikacij, ki se je do danes široko uveljavila v robotiki. Njegov glavni namen je olajšati razvoj sistema robota iz več komponent in načrtovanje le teh v generičnem slogu, torej da bodo z minimalnimi popravki delovale tudi na drugih robotih.

Projekt se je začel leta 2007 v Stanford Artificial Intelligence Laboratory (SAIL), nato pa so ga leta 2008 dokončno razvili v Willow Garage. V zadnjem

času ga vzdržuje Open Source Robotics Foundation (OSRF).

ROS skupnost izjemno narašča in ima člane s celega sveta, ki delijo svojo kodo, izkušnje in ugotovitve. Zato je večina večjih podjetji, ki se ukvarja z robotiko svoje projekte iz lastnih platform prenesla na ROS.

Razlogov, da se je ROS tako dobro uveljavil v svetu robotike, je poleg velike skupnosti še veliko. Vsebuje veliko funkcionalnosti, ki so pripravljene za uporabo, npr. SLAM (Simultaneous Localization and Mapping) in AMCL (Adaptive Monte Carlo Localization) programska paketa, ki se uporabljata pri avtonomni navigaciji robota in orodij, kot so `rqt_gui`, RViz in Gazebo [11]. ROS podpira veliko senzorjev in pogonskih naprav, saj pride zapakiran z njihovimi gonilniki (npr. Velodyne-LIDAR, Kinect itd.). Največja prednost so zagotovo dobre lastnosti platforme, kot so:

delovanje med platformami: ROS omogoča komunikacijo med različnimi vozlišči, ki so lahko programirana v kateremkoli jeziku, ki ima knjižnico za ROS odjemalca (C, C++, Python, Java),

modularnost: Problem pri monolitnih robotskih aplikacijah nastane, ko se nek proces sesuje ali ustavi in se ustavi celotna aplikacija. ROS s pisanjem vozlišč poskrbi za modularnost. Če se torej sesuje eno vozlišče, ni nujno, da se zaustavi celotna aplikacija,

sočasno obravnavanje virov: Pisanje kode za istočasno upravljanje dveh procesov (npr. zajemanje slik s kamero in podatkov s senzorjem LIDAR) ni nikoli zabavno. Poleg tega z naraščanjem števila procesov raste kompleksnost kode. ROS to težavo odpravi z gonilniki in temami, na katere se lahko poveže več različnih vozlišč.

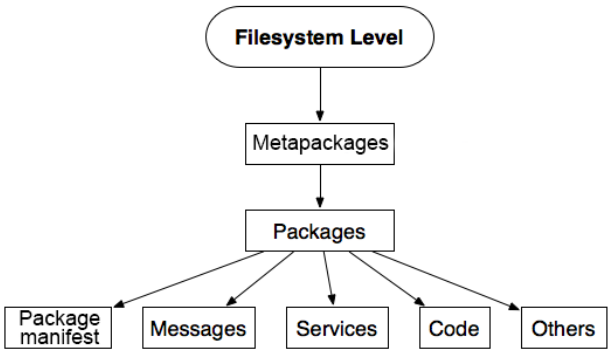
Kljub vsem dobrim lastnostim ROSa se nekateri ne odločijo zanj, saj je delo s platformo težavno. Navadno se osnove nauči hitro iz knjig in ROS wiki spletne strani, vendar je pot naprej do razvoja večjih aplikacij neprijetna, saj na spletu ni veliko vadnic oz. tečajev [10].

Druga težava se pojavi, ko želimo svojo aplikacijo simulirati. Glavni simulator za ROS je Gazebo, ki zahteva ogromno dela in konfiguriranja, saj je

vse potrebno narediti v ROSu (nima vgrajene možnosti programiranja). Poleg tega je težavno tudi modeliranje robota za simulacijo. Model je potrebno opisati v URDF (Unified Robot Description Format) formatu, ki temelji na XML.

V nadaljevanju bodo opisani glavni koncepti in arhitektura ROSa, ki je v osnovi razdeljena na tri nivoje:

1. **ROS datotečni sistem** je skupina konceptov, ki ponazarjajo, kako je ROS zgrajen in kaj vse potrebuje za delovanje. Datoteke v ROSu, so kot v operacijskih sistemih organizirane v določenem stilu tako, da zagotavlja centraliziran razvoj aplikacije, medtem ko omogoča dovolj fleksibilnosti in orodij za decentralizacijo odvisnosti (ang. dependencies) drugih knjižnic od projekta.



Slika 4.1: Datotečni sistem v ROSu. [11]

- **Paketi:** Programska koda v ROSu je organizirana v pakete, najosnovnejšo enoto, v kateri so vozlišča, knjižnice, konfiguracijske datoteke in ostalo, ki skupaj logično tvorijo modul. Dobra lastnost paketov je, da jih lahko uporabimo večkrat, kar zagotavlja modularnost.

  ROS ima svoja orodja, s katerimi kreiramo, spreminjamo in delamo s paketi. Nekatera glavna orodja so `catkin_create_pkg`, ki ga

uporabimo za kreiranje novega paketa, `rospack` za pridobivanje informacij paketa, `catkin_make` za gradnjo paketov v delovnem področju in `rosdep` za inštalacijo vseh odvisnosti sistema za paket.

- **Manifest paketa** je `package.xml` datoteka znotraj paketa, ki vsebuje informacije o avtorju, licenci, odvisnostih in druge podrobnosti paketa.
- **Meta paketi** vsebujejo le eno datoteko; `package.xml`. Uporabljajo se za referenco do drugih paketov, ki so običajno grupirani po podobni funkcionalnosti.
- **Meta manifest paketa** je podoben kot manifest paketa, le da lahko vsebuje še druge pakete kot odvisnosti.
- **Sporočila** se uporabljajo za komunikacijo med vozlišči. Definiramo jih v msg poddirektoriju znotraj paketa. V vsakem sporočilu so zapisani tipi podatkov, ki se bodo pošiljali. To so lahko primitivni tipi, ki jih podpira ROS, ali pa definiramo svoje sporočilo z uporabo standardnih.
- **Storitve** se prav tako uporabljajo za komunikacijo med vozlišči po principu zahtevek/odgovor, torej eno vozlišče pošlje zahtevek in čaka na odgovor. Prav tako moramo tip storitve definirati v paketu.

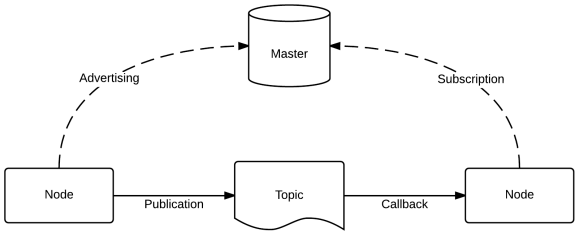
2. **Računski graf *(ang. Computation Graph level)***. Računski graf skrbi za P2P *(ang. Peer-to-Peer)* komunikacijsko omrežje med procesi, ki so predstavljeni kot eden glavnih konceptov: `vozlišče, tema, server parametrov, ROS vreča, ROS master, storitev` ali `sporočilo`.

   - **Vozlišče:** Na vozliščih se izvaja procesiranje podatkov. Vsa koda, ki je zadolžena za delovanje robota, je razdeljena po vozliščih. Njihov glavni namen je čim bolj poenostaviti sistem, zato želimo

vozlišča pisati tako, da opravljajo manj nalog in ne kot večji, kompleksnejši sistem.

Npr. če imamo vozlišče, ki skrbi za obdelavo slik iz kamere, podatkov iz senzorja LIDAR ter nadzoruje varnost robota, je bolj smiselno, da to vozlišče razdelimo na tri enostavnejša.

- **Tema** *(ang. Topic)* je komunikacijski program, kamor vozlišča pošiljajo oziroma objavljajo podatke ali pa se na njih naročijo in podatke dobivajo. Vsako vozlišče je lahko naročeno na katerokoli temo, četudi nanjo ne objavlja nobeno drugo vozlišče.
- **Glavno vozlišče** *(ang. Master)***:** skrbi za pravilno delovanje komunikacije med vozlišči. Vsakemu vozlišču omogoča vpogled do drugih in vzpostavitev povezave med dvema ter registracijo imena, ko je vozlišče ustvarjeno.



Slika 4.2: Komunikacija med vozliščema preko teme s pomočjo glavnega vozlišča. [12]

- **Strežnik za parametre** je del glavnega vozlišča in dovoljuje centralno shranjevanje podatkov, do katerih lahko dostopajo ali jih spreminjajo vsa vozlišča.
- **Vreče** so izjemno uporaben mehanizem shranjevanja podatkov o komunikaciji v robotu. V realnem okolju lahko zajamemo podatke vseh senzorjev in nato na njih razvijamo in izboljšujemo algoritme ali izvajamo teste.

3. **Skupnost ROS** koncept so sredstva, ki omogočajo sodelovanje med več skupnostmi (izmenjava programske kode in znanja). Sredstva v ROSu, ki to zagotavljajo, so razvijanje novih distribucij, ki olajšajo razvoj projektov, organizirano omrežje repozitorijev s kodo, tako da lahko različne institucije razvijajo svoje programske komponente ter ROS Wiki, sistem za prijavo hroščev, ROS answers forum in blog, ki pomagajo razvijalcem s težavami v kodi.

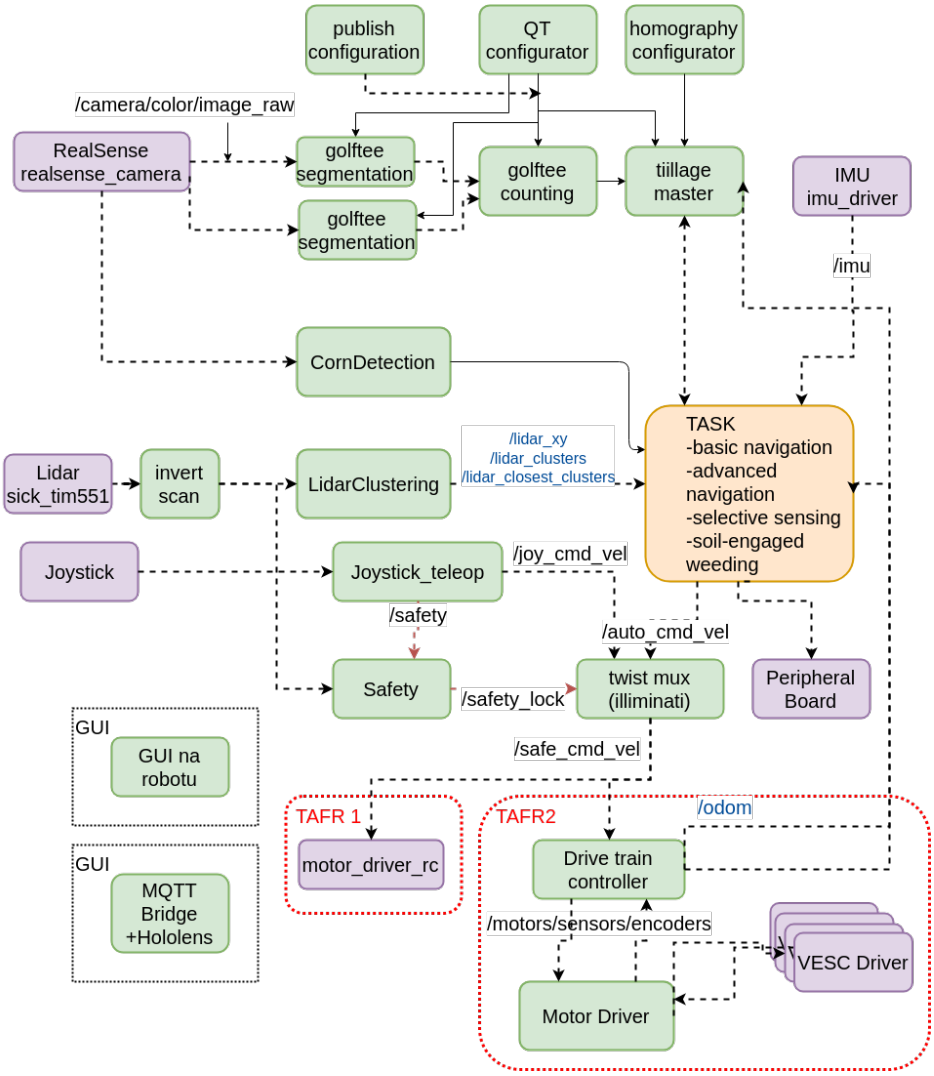
4.2 Aplikacija TAFR

Tafr aplikacija je skupek datotek, knjižnic in programske kode, ki skrbijo za celotno delovanje arhitekture robota – od nižjega (gonilniki, senzorji) do višjega (računalniški vid, vizualizacija, simuliranje) nivoja. Zgrajena je modularno, zato lahko poganja TAFR 1.0 in TAFR 2.0 robota.

V tem razdelku so opisana vozlišča, ki predstavljajo glavni prispevek te diplomske naloge in so odgovorna za avtonomno vožnjo robota po polju in povezave med njimi. V splošnem so to vozlišča, s katerimi smo rešili problem, zastavljen v diplomski nalogi in ki skrbijo za obdelavo podatkov iz senzorjev (LIDAR, odometrija, IMU, kamera) in del kontrolnega vozlišča (poimenovanega TAFR na sliki 4.3), v katerem se ti podatki v operacijah uporabijo za navigacijo. Na sliki 4.3 so predstavljena še ostala vozlišča, ki skrbijo za varnost, daljinsko upravljanje robota, zaznavanje modrih in rdečih podstavkov za golf žogice in mehanizem za pobiranje podstavkov.

4.2.1 Interpretacija senzorskih podatkov

Za lažjo predstavo podatkov, ki jih izmerijo senzorji robota, imajo robotski sistemi tipično več koordinatnih sistemov, ki se v času spreminjajo (npr. kje in kdaj smo izmerili podatke v odvisnosti od pozicije robota ali katerega drugega objekta). Običajno imajo enega za svet in vsako pomembnejšo komponento, ki sestavlja robota (npr. robotske roke, kolesa ...). Osrednji koordinatni sistem robota TAFR 2.0 je prikazan na sliki 4.4. Za nadzor



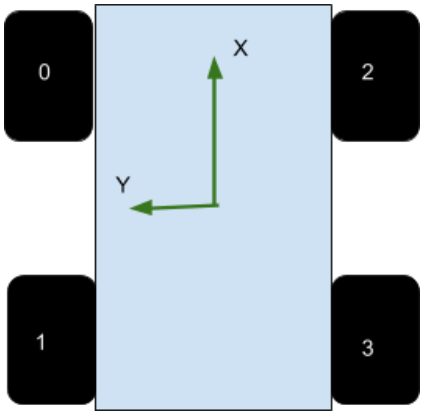
Slika 4.3: Zgradba aplikacije TAFR. Zeleni okvirji predstavljajo vozlišča, ki jih je napisala ekipa Tafr, vijolični okvirji nižjenivojske in odprtokodne rešitve, rumen okvir pa kontrolno vozlišče, ki implementira operacije za avtonomno navigacijo robota.

koordinatnih sistemov skrbi knjižnica *tf2*, s katero lahko dobimo podatke o odvisnosti dveh koordinatnih sistemov v določenem času, poziciji objekta v enem koordinatnemu sistemu relativno na drugega (npr. pozicija koruze v koordinatnemu sistemu za LIDAR glede na koordinatni sistem robota ali

sveta) in še veliko več.

Deluje lahko v distribuiranem sistemu, kar pomeni da lahko informacije o koordinatnih sistemih robota dobi katerakoli komponenta v ROSu ali v drugem primeru centralno, kar pomeni, da so koordinatni sistemi shranjeni na centralnem strežniku.

Knjižnica tf2 hrani vse koordinatne sisteme v drevesni strukturi, kot je prikazano na sliki 4.5, in omogoča transformacijo točk in vektorjev med njimi, s čimer razvijalcu prihrani veliko časa.

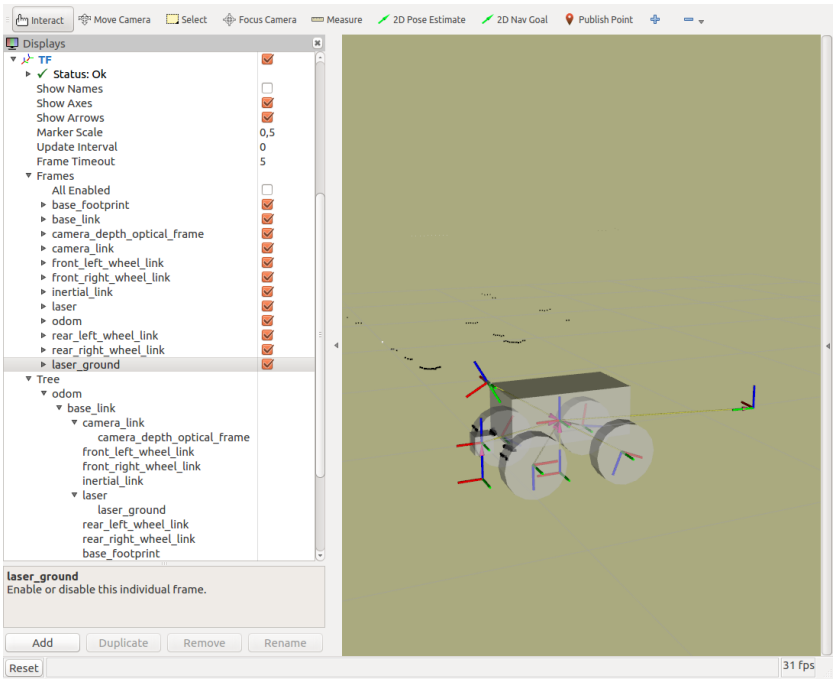


Slika 4.4: Koordinatni sistem robota in oznake koles. Os x kaže v smeri naprej, os y pa v smeri leve strani.

4.2.2 Priprava podatkov s senzorja LIDAR

Senzor LIDAR vrača podatke v obliki tabele točk, ki jih zazna v okolju. Ker je v našem primeru to polje koruze, lahko na sliki 4.6 vidimo, da imajo surovi podatki veliko šuma (razvejanost listov, odboji svetlobe itd.), zato jih je pred uporabo potrebno obdelati.

Glavna naloga vozlišča *LidarClustering* je obdelava LIDARskih podatkov in iskanje najbližje gruče na levi in desni strani robota. Zaradi velikega obsega podatkov, se najprej omejimo na manjšo množico, ki je v bližini robota. Zmanjšamo tudi kot, pod katerim so podatki zajeti, tako da ne zaznamo ko-

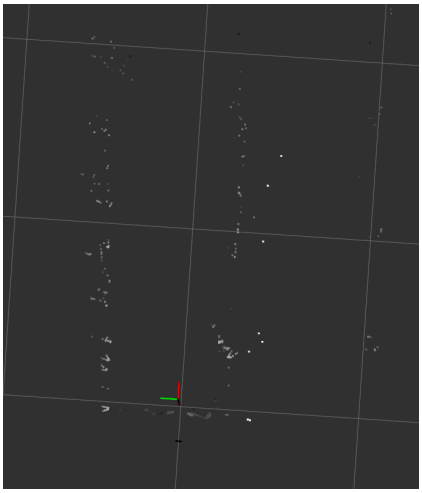


Slika 4.5: Prikaz koordinatnih sistemov robota in njihove drevesne strukture (v pogledu Displays). Na desni strani je koordinatni sistem za odometrijo, ki je povezan s koordinatnim sistemom *base_link* v sredini robota, na katerega so povezani še ostali.

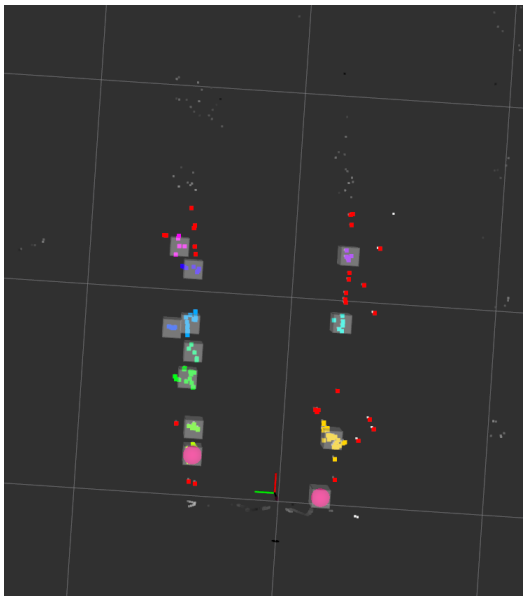
les robota. Vse ustrezne točke pretvorimo iz polarnih koordinat v kartezične in jih uredimo po evklidski razdalji. Na seznamu shranjenih točk nato izvedemo algoritem za gručenje DBSCAN, ki je opisan v 3.1. Izračunamo še centre dobljenih gruč in iz njih najbližjo gručo na levi in desni strani, ter jih pretvorimo v ROSov oblak točk *(ang. ROS pointcloud)*.

4.2.3 Zaznavanje koruze

Glavna naloga vozlišča je prepoznavanje koruze iz slike. Ker so včasih drugi senzorski podatki nenatančni, si pri zaznavanju vrst lahko pomagamo s kamero. Zaznavanje koruze iz slik še ni vključeno v glavno aplikacijo, ker robot nima dovolj računske moči.



Slika 4.6: Surovi podatki, zajeti s senzorjem LIDAR med vrstami koruze.



Slika 4.7: Rezultat gručenja na podatkih iz slike 4.6.

Za izbiro pravega segmenta slike si pomagamo s senzorjem LIDAR. Ko zaznamo novo vrsto s senzorjem in smo približno ob njej, zajamemo sliko široko `w` in visoko `h`. Velikost izrezane slike prilagajamo v odvisnosti od

vidnega kota (pozicije) kamere ter velikosti koruze. Če je kamera usmerjena bolj navzdol, moramo poskrbeti, da odrežemo del poti pred koruzo. Če je usmerjena bolj navzgor, moramo odrezati ostali del vrste koruze (in ozadje za vrsto, če ga zaznamo). Na učnih slikah smo zajeli izrezek širok 260 slikovnih elementov (130 slikovnih elementov v vsako smer od središča slike) in odrezali zgornjih 10 slikovnih elementov ter spodnjih 120 slikovnih elementov (da smo izločili kolo in pot). Primer dobrega primera rezanja slike je prikazan na sliki 4.9.

Slika 4.8: Originalna slika zajeta, ko s senzorjem LIDAR zaznamo da smo ob koruzi.

4.2.4 Operacije za avtonomno navigacijo

V kontrolnem vozlišču (oziroma vozlišču Task na diagramu 4.3) se vsi obdelani podatki uporabijo za avtonomno navigacijo robota po polju med vrstami koruze. V vozlišču so napisane vse funkcije, ki skrbijo za vožnjo in upravljanje robota. Trenutna verzija aplikacije poleg navigacije podpira še zaznavanje in pobiranje modrih in rdečih žebljičkov (podstavkov) za žogice za golf s pomočjo mehanizma Tilda (tillage master paket iz diagrama 4.3).

Slika 4.9: Rezultat rezanja slike.

Kontrolno vozlišče ima implementiranih več objektov, ki skrbijo za nemoteno delovanje robota. Objekt *RobotHandler* je zadolžen za pridobivanje in objavljanje podatkov iz in na teme, vizualizacijo podatkov v *rvizu*, daljinsko upravljanje ter skrbi za regulator PID in nastavitve senzorjev. Objekt *RobotOperation* vsebuje operacije za avtonomno vožnjo in si pomaga z objektoma *CornFollow*, ki je zadolžen za računanje napak v vožnji in *RowTracker*, ki je zadolžen za štetje vrst koruze.

RowTracker objekt vsebuje algoritem za štetje vrst z napovedovanjem iz poznanih podatkov o polju. Idejo delovanja predstavlja algoritem 1. Robot v prvi iteraciji določi, kje je naslednja vrsta. To stori tako, da vrednost napove s pomočjo parametrov ali pa jo enostavno nastavi na vrednost najbližje gruče (ki jo dobi iz vozlišča *LidarClustering*). V naslednjih iteracijah robot napoved posodablja z novejšimi odčitki senzorja LIDAR in preverja ali jo je prečkal. To stori tako, da primerja koordinati osi x svoje pozicije in napovedane vrste. Če sta si dovolj blizu sklepa, da je ob napovedani vrsti, zato poveča število preštetih vrst in napove naslednjo.

Druga glavna operacija je sledenje koruzi znotraj vrste (*RowFollow*), ki

robota usmerja tako, da z računanjem napake med vožnjo (odmaknjenost robota od sredine poti) motorjem ustrezno nastavlja hitrost. Robot se v vsaki iteraciji odloči, ali bo sledil levi ali desni vrsti. Odloči se na podlagi najbližjih gruč, ki jih vrne vozlišče *LidarClustering*. Če je bila zaznana gruča le na eni strani, se odloči za tisto, sicer za bližjo. V naslednjem koraku robot preveri, ali je že na koncu vrste. Če ni zaznal nobene gruče v določenem rangu, se operacija zaključi. Za čim bolj gladko vožnjo med vrstama robot v vsaki iteraciji s pomočjo objekta *CornFollow* izračuna napako odmika od sredine poti in iz napake hitrost levih in desnih koles. S tem poskrbimo, da je robot med navigacijo čim bolj enakomerno odmaknjen od leve in desne vrste koruze.

Ostale implementirane operacije so še *spotRotation*, *inRowDrive* ter *outRowDrive*. *spotRotation* na mestu obrne robota za 90° v želeno smer. Iz podatkov o orientaciji, ki jih robot dobi iz senzorja IMU, izračunamo kot, pod katerim mora biti robot po končani operaciji. Funkcija *inRowDrive* navigira robota na pot, *outRowDrive* pa za izvoz iz poti.

Implementira tudi glavno funkcijo, ki ji podamo vektor ukazov tipa (stran, št. vrst) po katerih se mora robot navigirati po polju. Avtonomna navigacija je sestavljena iz opisanih operacij, kot prikazuje algoritem 2.

Algoritem 1: Preprost algoritem za napovedovanje vrst iz znanih podatkov o polju s potrjevanjem podatkov iz senzorja LIDAR. Vsaka funkcija, ki se pravilno izvede, vrne vrednost 1, sicer 0.

```
while prestete_vrste < stevilo_vrst do
    if zacetek then
        if zaznana gruča z LIDAR then
            napovedana_gruca = zaznana_gruca;
        else
            napovej gručo(napovedana_gruca);
    if zaznana gruča z LIDAR v dovoljeni razdalji then
        posodobi(napovedana_gruca);
    if razdalja(napovedana_gruca, pozicija_robota) ¡ epsilon) then
        prestete_vrste + 1;
        napovej gručo(napovedana_gruca);
```

Algoritem 2: Vrstni red operacij, v katerem se izvaja avtonomna navigacija robota.

```
switch kazalec_operacij do
    case 1 do
        kazalec_operacij += inRowDrive()
    case 2 do
        kazalec_operacij += rowFollow()
    case 3 do
        kazalec_operacij += outRowDrive()
    case 4 do
        kazalec_operacij += spotRotation(90 * strain)
    case 5 do
        kazalec_operacij += perpendicularRowFollow(st_vrst, stran)
    case 6 do
        kazalec_operacij += spotRotation(90 * strain)
    case 7 do
        kazalec_operacij = 1;
        naslednji_ukaz()
```

Poglavje 5

Eksperimentalni rezultati

5.1 Zajemanje podatkov

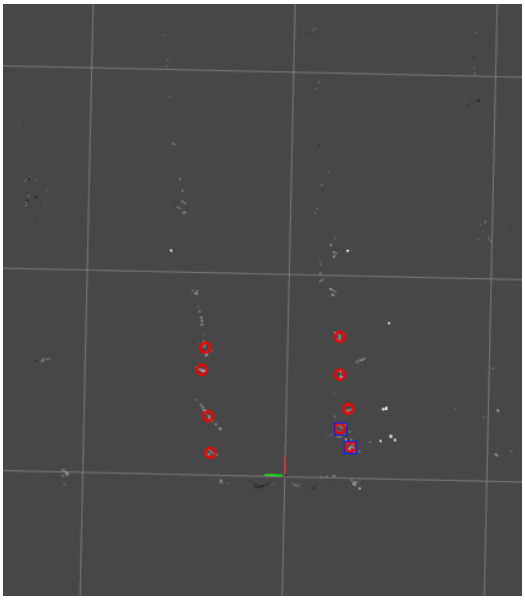
Podatke smo zajemali na štirih različnih poljih z različno velikostjo koruze. Tekmovanje FRE zagotavlja, da so vrste koruze posajene v razmiku približno 75 cm, pot pa je dolga 15 m. Vse rastline koruze so visoke med 20 cm in 50 cm. Podatki so zajeti med daljinskim upravljanjem in avtonomnim delovanjem robota in so v obliki ROS vreče, ki ima za vsako časovno iteracijo shranjeno celotno komunikacijo med temami in vozlišči ter pridobljene podatke (LIDAR, kamera, IMU, odometrija ...). Nekaj težav smo imeli s shranjevanjem podatkov, saj se je zaradi premajhne kapacitete disk hitro zapolnil, velikost vreč pa je velika, ker shranjujemo tudi slike.

5.2 Rezultati zaznavanja koruze s senzorjem LIDAR

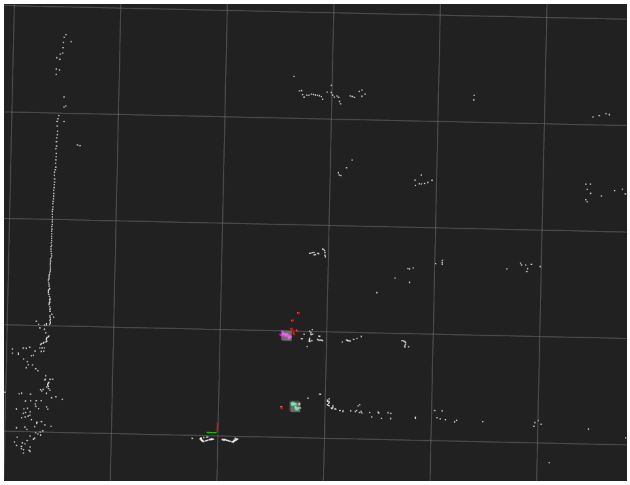
Med razvojem robota smo vožnjo sprva testirali v simulatorju, nato pa še na umetni koruzi, zgrajeni iz lesenih palčk (slika 5.1). Tako okolje je seveda bolj idealno za delovanje algoritmov kot realno polje, saj je šuma v podatkih veliko manj. Algoritem DBSCAN je zato zaznaval koruzo veliko bolje kot kasneje na testiranju. Med testiranjem smo zato morali parametre algoritma

prirejati. V več iteracijah smo z različnimi kombinacijami parametrov dobili rezultate gručenja, ki smo jih nato primerjali med sabo in z dejanskimi rezultati. Primer neustreznih parametrov je na sliki 5.2, kjer smo zahtevali preveliko število točk znotraj krožnice. Z modrim kvadratkom je označena zaznana koruza z gručenjem; rdeči krogci predstavljajo približne pozicije dejanskih rastlin koruze. Prav tako ni v redu, če s parametri zahtevamo premajhno število točk znotraj krožnice, saj lahko tako kot koruzo zaznamo tudi šum. Primera dobrega gručenja z dobro nastavljenimi parametri sta na sliki 5.4 in 5.3.

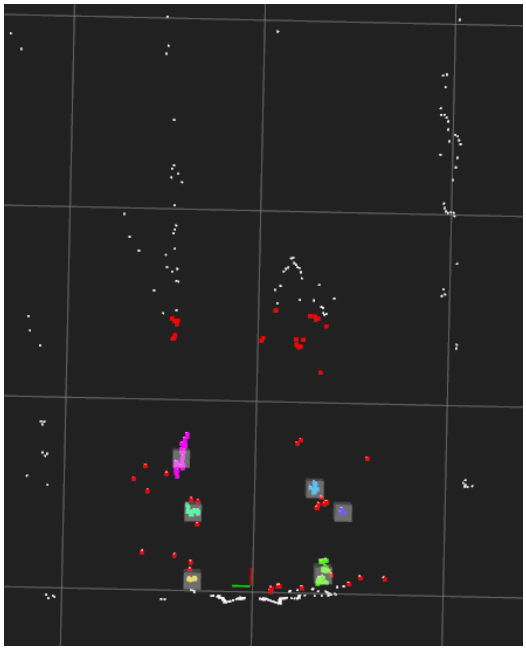
Slika 5.1: Testiranje na umetni koruzi.



Slika 5.2: Primer slabe izbire parametra, ko zahtevamo preveliko število točk znotraj krožnice. Posledično algoritem zazna le zelo nasičene predele. Z rdečim krogcem so označene dejanske koruze, z modrim kvadratkom pa zaznane z algoritmom.



Slika 5.3: Primer dobrega gručenja, medtem ko robot šteje vrste.



Slika 5.4: Primer dobrega gručenja, kjer je robot na poti med vrstami koruze.

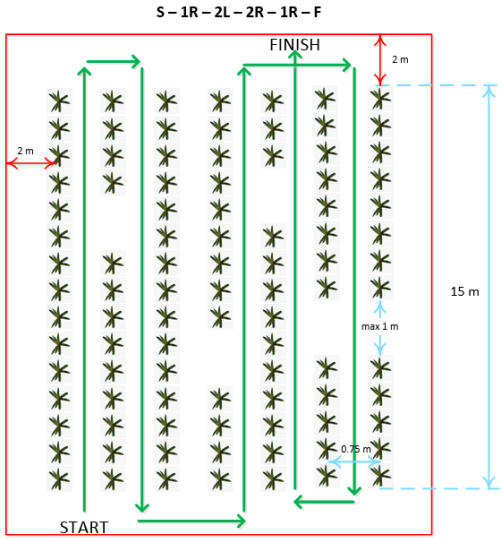
5.3 Rezultati s tekmovanja

Na tekmovanju smo zasedli skupno 10. mesto med 15 ekipami. Pred tekmovanjem smo pričakovali boljše rezultate, vendar smo zaradi težav z elektroniko morali nastopiti z robotom TAFR 1.0. Čeprav imata oba robota iste senzorje, lahko sklepamo, da bi se robot TAFR 2.0 na tekmovanju bolje odrezal, saj je njegova mehanska zasnova veliko boljša. Robot TAFR 1.0 nima ustreznega vzmetenja, zato je bil oprijem koles slabši; prav tako so bili odčitki s senzorja LIDAR zaradi stalnega nagibanja manj natančni.

Prva naloga je bila enostavna navigacija robota po polju koruze. Robot je moral peljati mimo čim več zaporednih ukrivljenih vrst v treh minutah. Pri tej nalogi smo imeli slabe rezultate zaradi pozicije senzorja LIDAR na robotu. Pritrjen je bil previsoko, zato je zaradi razvejanih listov koruze zaznaval preveč šuma v podatkih. Zaradi šuma je robot ves čas zaznaval ovire na poti in se velikokrat ustavil, zato ga je bilo potrebno večkrat ponovno

zagnati. Prva nastavitev senzorja je na testnem polju delovala dobro, saj so bile koruze malo bolj narazen in manj razvejane, kot na tekmovalnem polju. Po prvi nalogi smo senzor LIDAR premaknili približno 5cm nižje. Robota smo ponovno peljali na tekmovalno polje, kjer se je odrezal veliko bolje.

Druga naloga je bila napredna navigacija. Naloga je potekala na drugem tekmovalnem polju, kjer so bile vrste z manjkajočimi koruzami ravne, vendar je bila podlaga bolj groba z veliko kamenja na poteh. Nižja postavitev senzorja LIDAR je prispevala k veliko boljšemu rezultatu kot v prvi nalogi, saj je bil postavljen v višino stebel koruze in ne listov (zaznaval je manj šuma). Robotu je uspelo priti do zadnjega ovinka. Torej prevozil je vse razen zadnje poti. V tej nalogi smo osvojili tretje mesto.



Slika 5.5: Navodila za napredno navigacijo. [9]

V tretji nalogi (slika 5.6) smo morali na poteh zaznati plevel predstavljen z modrimi in rdečimi podstavki za golf žogice. Če je bilo več rdečih podstavkov je moral robot potrobiti enkrat, sicer pa dvakrat. Zaznavanje plevela s kamero ni bilo učinkovito zaradi neustrezne zaščite proti sončni svetlobi. Robotu je uspelo pravilno zaznati plevel le enkrat od devetih možnosti. V tej nalogi so se pojavili problemi z elektroniko robota, zato je večkrat prenehal

delovati.

Pri četrti nalogi je bil plevel v obliki rdečih podstavkov na pot postavljen trikrat. Podstavke je bilo potrebno zaznati in jih z mehanizmom za pobiranje pobrati. Robotu je uspelo pobrati dva podstavka, nato pa se je na koncu prve poti zaradi težav z elektroniko ustavil.

Operacije, ki skrbijo za avtonomno navigacijo robota po polju koruze so zaradi dobrih nastavitev parametrov med testiranjem delovale zelo dobro. Dobro je deloval tudi algoritem za štetje vrst koruze, saj ni zgrešil niti ene.

Slika 5.6: Robot TAFR 1.0 na tekmovanju FRE v tretji nalogi.

5.4 Rezultati prepoznavanja s kamero

Model za prepoznavanje je bil naučen na podatkovni množici 720 barvnih slik velikosti 50×50 slikovnih elementov (339 slik koruze in 381 slik poti). Učenje, kot vidimo na sliki 5.9, je potekalo v 60 *epohah*. To pomeni, da je bila cela podatkovna množica poslana po mreži z arhitekturo, opisano v 3.7 šestdesetkrat. Od približno 60. in do 200. epohe se natančnost modela ni

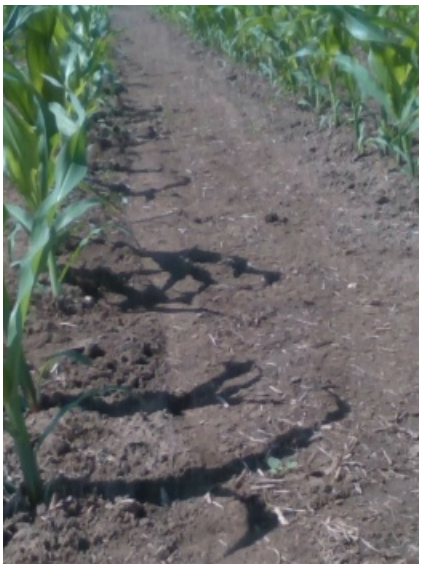
povečala. Testiran je bil na 168 tesnih slikah s približno 92 % uspešnostjo, kar pomeni, da je pravilno klasificiral 155 testnih slik (evalvacija modela, kot vidimo v zadnji vrstici na sliki 5.9 vrne približno 8 % napako).

Ker robot nima dovolj računske moči, zaznavanje s kamero še ni del sistema, zato je bil testiran le na domačem računalniku in predstavlja smernice za nadaljnje delo.

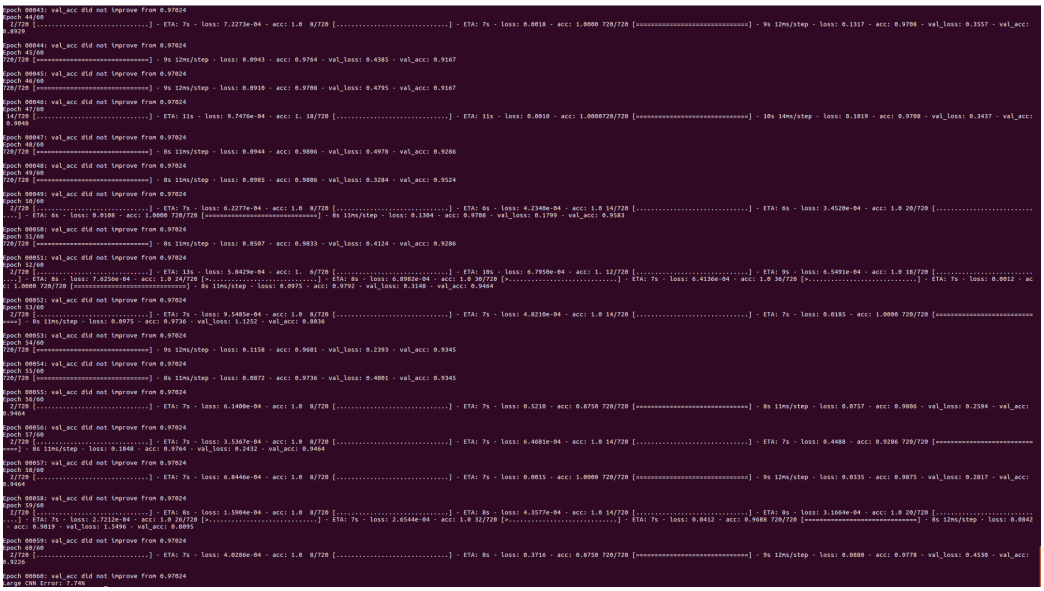
Za pravilno delovanje moramo modelu podati izrez slike na katerem je koruza lepo vidna. Pri tem si pomagamo s senzorjem LIDAR tako, da ko zaznamo koruzo poleg robota, zajamemo osrednji del slike. Če pa je v podatkih nekaj šuma, lahko zajamemo napačen del slike za prepoznavo, kakor lahko vidimo na sliki 5.8, ki je izrez originalne slike 5.7.

Metodo zaznavanja bi lahko izboljšali tako, da zajamemo nekoliko več slik, ko smo ob vrsti. Slike bi nato ustrezno rezali in imeli več možnosti za dober izrez. Prav tako bo potrebno testirati, kako rezanje deluje glede na odmaknjenost robota od polja. Če bi slike zajemali nekoliko bolj oddaljeni od polja, bi zajeli več informacij in bi jih lahko mogoče bolj natančno rezali. Implementacijo bi lahko izboljšali s kombiniranjem informacije iz slike in 3D oblaka točk.

Slika 5.7: Originalna slika, ki smo jo zajeli ko je senzor LIDAR zaznal da smo ob vrsti.



Slika 5.8: Primer slabega rezanja slike zaradi šuma v podatkih.



Slika 5.9: Učenje modela v 60 korakih z 92% natančnostjo.

Poglavje 6

Zaključek

V diplomski nalogi smo reševali problem avtonomne navigacije robota po polju za tekmovanje FRE. Načrtovali in razvili smo svojega robota in ga opremili z različnimi senzorji. Robota smo v celoti sestavili sami iz ročno izdelanih in kupljenih delov. Razvili smo tudi aplikacijo, ki skrbi za delovanje robota in ga avtonomno navigira po polju. Aplikacija je izdelana modularno, tako da lahko poganja več podobnih robotov z istim senzorji.

Za avtonomno vožnjo robota smo implementirali dele aplikacije, ki skrbijo za obdelavo podatkov, ki jih izmeri senzor LIDAR, in operacije, ki izvajajo avtonomno navigacijo, kot so sledenje koruzi ob poti in štetje vrst koruze z napovedovanjem iz poznanih podatkov. Ker pogoji niso vedno tako dobri kot na tekmovanju in je lahko koruza bolj razvejana in je okoli stebel koruze plevel, si s senzorjem LIDAR pri štetju vrst ne moremo pomagati. Zato smo naučili model za prepoznavanje koruze iz kamere. Zaradi pomanjkanja računalniške moči modela ne moramo testirati na robotu ampak samo na računalniku. Zaznavanje deluje če dobro izrežemo del slike, kar ocenimo s senzorjem LIDAR.

Možnosti za izboljšave je še veliko. Izboljšali bi lahko rezanje slike pri zaznavanju koruze, saj trenutna aplikacija zajame le eno sliko, kjer privzamemo, da je koruza na sredini in moramo le pravilno odrezati ostale dele slike. To bi lahko izboljšali tako, da zajamemo več slik in na njih izvajamo

prepoznavo. V primeru dovolj računalniške moči bi lahko uporabili algoritem Mask R-CNN, ki učinkovito zaznava objekte iz slik. V tem primeru bi veliko dela zahtevalo pridobivanje in označevanje velike količine slik. V samo vožnjo, kjer se osredotočamo na podatke iz senzorja LIDAR, bi lahko vpeljali varnostne mehanizme z merjenjem časa in računanjem prevožene razdalje z odometrijo. Ker poznamo dimenzije polja, bi lahko to izkoristili in uporabili v operacijah. Avtonomno vožnjo bi lahko izboljšali tudi s senzorjem GPS, ki je sicer na tekmovanju prepovedan, ali boljšim 3D senzorjem LIDAR, ki zelo veliko stane.

Načrtov za prihodnost je torej veliko, vendar so odvisni od sredstev, ki jih bomo uspeli dobiti. Prvi korak bo pridobitev dodatne računske moči (verjetno v obliki strežnika, na katerega bo robot povezan), da bomo lahko začeli razvijati vožnjo s pomočjo kamer – nekoč mogoče le s kamerami.

Literatura

[1] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. pages 1–33, 2012.

[2] CNN. Dosegljivo: https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050, 2018. [Dostopljeno: 17. 8. 2018].

[3] CNN. Dosegljivo: https://chatbotslife.com/how-neural-networks-work-ff4c7ad371f7, 2018. [Dostopljeno: 17. 8. 2018].

[4] CNN. Dosegljivo: https://www.researchgate.net/figure/A-single-ANN-neuron-with-its-elements_fig5_285400200, 2018. [Dostopljeno: 17. 8. 2018].

[5] Nivoji CNN. Dosegljivo: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2, 2018. [Dostopljeno: 20. 8. 2018].

[6] DBSCAN. Dosegljivo: https://www.quora.com/How-does-DBSCAN-algorithm-work, 2018. [Dostopljeno: 24. 8. 2018].

[7] Pravilo delta. Dosegljivo: https://en.wikipedia.org/wiki/Delta_rule, 2018. [Dostopljeno: 20. 8. 2018].

[8] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231, 1996.

[9] Field Robot Event. Dosegljivo: http://www.fieldrobot.com/event/index.php/tasks/task-2/, 2018. [Dostopljeno: 15. 8. 2018].

[10] Lentin Joseph. *Mastering ROS for Robotics Programming*. Packt publishing, 2015.

[11] Fernandez Mahtani, Sanchez. *Effective Robotics Programming with ROS Third Edition*. Packt publishing, 2016.

[12] ROS. Dosegljivo: https://fr.wikipedia.org/wiki/Robot_Operating_System, 2018. [Dostopljeno: 15. 8. 2018].

[13] Jianxin Wu. *Introduction to Convolutional Neural Networks*. Lamda group, 2017.

[14] Geoffrey Hinton Yann LeCun, Yoshua Bengio. Deep learning. nature, 521(7553):436. *Nature*, pages 1–9, 2015.