Modeliranje okoline mobilnog robota lidarom

Horvat, Nikolai Ivan

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Nikolai Ivan Horvat

Zagreb, 2021.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

MODELIRANJE OKOLINE MOBILNOG ROBOTA LIDAROM

Mentor:
Prof. dr. sc. Mladen Crneković

Student:
Nikolai Ivan Horvat

Zagreb, 2021.

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

Zahvaljujem se mentoru prof. dr. sc. Mladenu Crnekoviću na iskazanom povjerenju, vodstvu i korisnim savjetima tijekom izrade rada, te prijateljima i kolegama na korisnim savjetima.

Na kraju, zahvaljujem se svojoj obitelji na pruženoj ljubavi, povjerenju i podršci tijekom cijelog studija.

Nikolai Ivan Horvat

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

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

ZAVRŠNI ZADATAK

Student: **NIKOLAI IVAN HORVAT** Mat. br.: 0035213869

Naslov rada na hrvatskom jeziku: **MODELIRANJE OKOLINE MOBILNOG ROBOTA LIDAROM**

Naslov rada na engleskom jeziku: **MODELING THE ENVIRONMENT OF A MOBILE ROBOT BY LIDAR**

Opis zadatka:

Planiranje gibanja mobilnog robota u nepoznatoj okolini u najvećoj mjeri ovisi o razini informacija o stanju okoline. LIDAR daje precizne informacije o smjeru i udaljenosti do prepreka iz čega treba izlučiti informacije više razine kao što su: najbliža prepreka, najveća prepreka, najširi prolaz itd. Koristiti Slamtec RPLIDAR A2M8. U radu je potrebno:

- prikupiti informacije koje daje LIDAR,
- prikazati podatke u polarnom dijagramu,
- definirati i izlučiti informacije više razine, te ih prikazati na zaslonu.

Potrebno je navesti korištenu literaturu i ostale izvore informacija, te eventualno dobivenu pomoć.

Zadatak zadan:
30. studenoga 2020.

Zadatak zadao:

Prof. dr. sc. Mladen Crneković

Datum predaje rada:
1. rok: 18 veljače 2021.
2. rok (izvanredni): 5. srpnja 2021.
3. rok: 23. rujna 2021.

Predviđeni datumi obrane:
1. rok: 22.2. – 26.2.2021.
2. rok (izvanredni): 9.7.2021.
3. rok: 27.9. – 1.10.2021.

Predsjednik Povjerenstva:

Prof. dr. sc. Branko Bauer

SADRŽAJ

POPIS SLIKA

POPIS OZNAKA

Oznaka	Jedinica	Opis
c	m/s	brzina svjetlosti - cca. 300 000 km/s
d	mm	pređena udaljenost laserske zrake u milimetrima
$height$	px	visina ekrana
kut_s	°	kut u stupnjevima
kut_s	rad	kut u radijanima
max_d	mm	maksimalna udaljenost skeniranja
q	mm	udaljenost točke 1 manje r_{2y}
r_1	mm	udaljenost točke 1
r_2	mm	udaljenost točke 2
t	s	vrijeme putovanja laserske zrake u sekundama
$udaljenost$	mm	udaljenost točke od ishodišta u polarnom dijagramu
$width$	px	širina ekrana
x		udaljenost točke u smjeru *x*-osi
y		udaljenost točke u smjeru *y*-osi
z	mm	udaljenost dviju točaka

SAŽETAK

Cilj ovog rada je proučiti način rada LiDAR-a i njegovu moguću primjenu u sklopu mobilnog robota čija se okolina modelira LiDAR-om.

U navedenom je zadatku potrebno prikupiti precizne podatke o smjeru i udaljenosti od prepreka, koje nam daje LiDAR, iz kojih će se izlučiti informacije više razine kao što su: najbliža prepreka, najveća prepreka, najširi prolaz itd. Podaci će biti prikazani u polarnom dijagramu i obrađeni pomoću mikro-računala. Napravit će se pregled svih tehnologija za razvoj takvog sustava.

Ključne riječi: LiDAR, Slamtec RPLidar A2M8, Raspberry Pi, Python

SUMMARY

The aim of this paper is to study the way the LiDAR scanner works and its possibilities as a mobile robot navigation system.

In this task, it is necessary to collect the precise information about the direction and distance to obstacles given by the LiDAR, from which higher-level information will be extracted, such as: the nearest obstacle, the largest obstacle, the widest passage, etc. The data will be shown in a polar diagram and will be processed using a micro-computer. An overview of all technologies for the development of such a system will be made.

Key words: LiDAR, Slamtec RPLidar A2M8, Raspberry Pi, Python

1. UVOD

Izrazitim napredovanjem mobilnih robota, kao i njihovom primjenom, rasla je i potreba za preciznijim opisom okoline za izbjegavanje prepreka što je rezultiralo povećanom primjenom i razvojem samog LiDAR-a čije je korištenje najzastupljenije u skeniranju različitih ekosustava.

LiDAR (eng. *Light Detection and Ranging*) skener uređaj je kojem je zadatak mjerenje udaljenosti između neke ciljne površine i samog uređaja. Također se naziva i optičkim - odnosno - laserskim radarom.

Tema ovog završnog rada modeliranje je okoline mobilnog robota pomoću LiDAR skenera zbog toga što planiranje gibanja mobilnog robota u nepoznatoj okolini u najvećoj mjeri ovisi o razini informacija o stanju okoline. LiDAR daje precizne informacije o smjeru i udaljenosti od prepreka iz kojih će biti izlučene informacije više razine poput najveće i najbliže prepreke, najšireg prolaza i sl. Korišteni su Slamtec RPLidar A2M8 te mikro-računalo Raspberry Pi za obradu primljenih prostornih podataka, koji će biti izlučeni na ekranu, u obliku polarnog dijagrama.

2. MOBILNI ROBOT

Mobilni robot primjer je robota koji se može kretati u predviđenoj okolini i nije fiksiran na jednom fizičkom mjestu. Može biti autonoman (AMR[1]) što znači da je sposoban upravljati nekontroliranim okolišem - bez potrebe za fizičkim ili elektro-mehaničkim uređajima za vođenje. Osim toga, mobilni roboti mogu se koristiti uređajima za vođenje koji im omogućavaju da putuju po unaprijed definiranoj trajektoriji u relativno kontroliranom okolišu (AGV[2]). Nasuprot tome, industrijski su roboti uglavnom stacionarni, no mobilni roboti postali su uobičajeniji, kako u komercijalnim, tako i u industrijskim okruženjima. Suvremene bolnice već dugi niz godina koriste autonomne mobilne robote za premještanje opreme. Skladišta ih koriste za učinkoviti transport robe s polica za skladištenje u zonu narudžbe, itd. Mobilni roboti također su glavni fokus trenutnih istraživanja, stoga gotovo svaki veći fakultet ima jedan ili više laboratorija fokusiranih na njihovim istraživanjima. [1]

Raspodjela mobilnih robota:

- Prema okolišu u kojem se koriste
- Prema načinu kretanja (kotači, gusjenice ili noge)
- Prema autonomnosti kretanja
    - Ručno upravljani robot pod nadzorom je vozača koji upravlja pomoću joysticka ili drugog upravljačkog uređaja.
    - Autonomni robot
        - Robot sa slučajnim smjerom kretanja u pravilu se odbija od prepreka
        - Potpuno autonoman robot sa sustavima prepoznavanja okoline

Za bilo koji mobilni robot važna je sposobnost navigacije. Navigacija robota znači sposobnost robota da odredi vlastiti položaj u svom referentnom okolišu, a zatim mogućnost planiranja puta prema nekom ciljanom mjestu. Potrebno je izbjeći opasne situacije poput sudara i nesigurnih uvjeta (temperatura, zračenje, izloženost vremenu itd.) , osim ako je

[1] Eng. Autonomous Mobile Robot
[2] Eng. Autonomous Guided Vehicle

neizbježno da prolazi kroz njih za ostvarenje njegove zadaće. Da bi se kretao u svom okruženju, robot ili bilo koji drugi autonomni uređaj zahtjeva izgled okoliša i sposobnost njegove interpretacije. [2]

Sliku okoliša mobilnog robota mogu pružiti uređaji poput ultrazvučnih senzora daljine, kamera, kontaktnih senzora, GPS-a, LiDAR-a, …

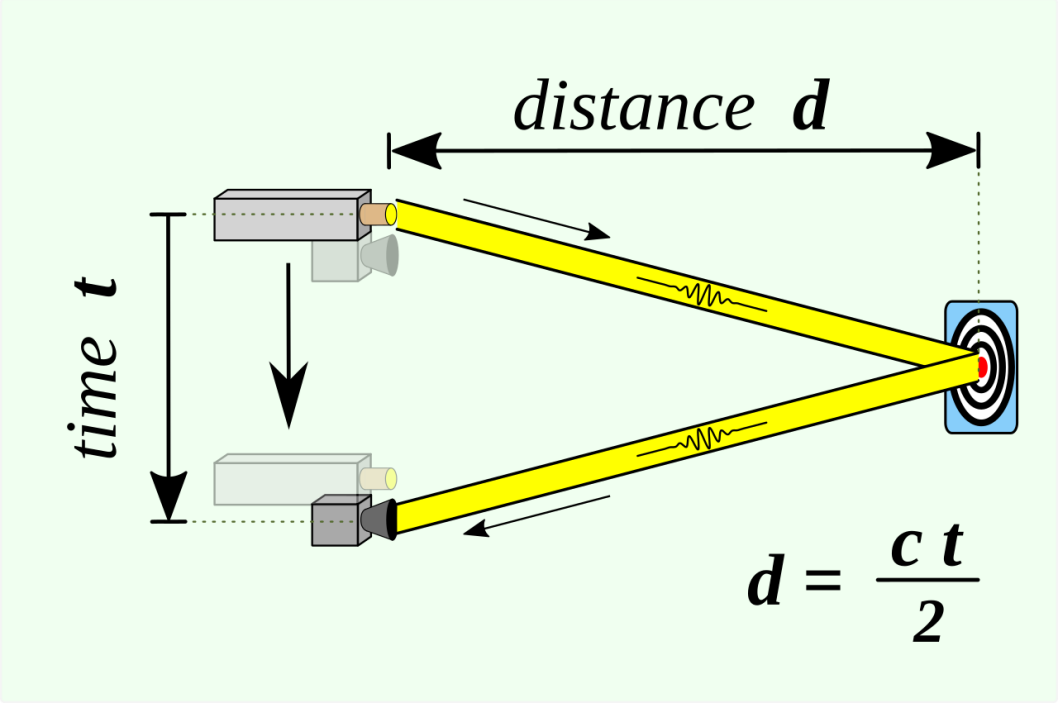
Slika 1: Mobilni robot FSB-a eMIR

3. LIDAR

LiDAR (eng. *Light Detection and Ranging*) je potpuno automatiziran, aktivan, optičko-mehanički postupak prikupljanja prostornih podataka dostupnih s aktualnog mjesta snimanja. Njihova je prednost prikupljanje niza vrlo preciznih podataka na većim udaljenostima. [3]

Navedena se udaljenost dobije određivanjem proteklog vremena između emisije kratkotrajnog laserskog[3] impulsa i dolaska refleksije tog impulsa (povratnog signala) na prijemnik senzora. Množenjem ovog vremenskog intervala s brzinom svjetlosti dobiva se prijeđena udaljenost, a rezultat dijeljenja te prijeđene udaljenosti s 2 udaljenost je između senzora i cilja. Jednadžba za izračun udaljenosti glasi:

$$d = \frac{c \cdot t}{2} \tag{1}$$



Slika 2: Vizualizacija izračuna udaljenosti

[3] Laser (eng. *Light Amplification by Stimulated Emission of Radiation*)

Ključne su razlike različitih LiDAR senzora valna duljina laserske zrake, snaga lasera, trajanje impulsa i brzina ponavljanja, veličina snopa i kut divergencije[4], određene specifičnosti mehanizma skeniranja (ako ih ima)… [4]

Podjela LiDAR skenera:

- Prema orijentaciji – nadir (točka ispod promatrača), zenit (točka iznad promatrača) ili bočno
- Prema mehanizmu skeniranja
    - Standardni tip skeniranja – okretanjem oko osi daje pogled od 360°
    - Solid-state LiDAR – fiksno vidno polje bez pokretnih dijelova
    - Flash LiDAR – širi bljesak svjetlosti preko velikog vidnog polja
- Prema okolišu primjene
    - Zračni – tijekom leta pričvršćen na zrakoplov i stvara 3D model krajolika koji služe za proučavanje šuma, rijeka, potoka, …
    - Kopneni – stacionarni ili pokretni
        - Mobilni LiDAR – dva ili više skenera pričvršćena na pokretno vozilo za prikupljanje podataka duž putanje (npr. izmjera ulica)

Moguća područja primjene LiDAR skenera: agrikultura, arheologija, biologija, geologija, rudarenje, fizika, astronomija, meteorologija, autonomna vozila, robotika i mnoga druga. [4]

U ovom je završnom radu korišten RPLidar A2M8 tvtke Shanghai Slamtec.Co, Ltd. standardnog mehanizma bočnog skeniranja za primjenu na kopnu.

[4] Kut skrenute zrake svjetlosti u odnosu na referentni pravac

3.1. RPLidar A2M8

RPLidar A2M8 (Slika 3) nova je generacija cijenom pristupačnih 2D LiDAR skenera s krugom skeniranja od 360° tvrtke Slamtec. Uređaj je u mogućnosti prikupiti 4000 uzoraka u sekundi ukoliko je brzina okretanja dovoljno velika, dok je domet 12 metara. Dobiveni se podaci mogu koristiti u modeliranju okoline, mapiranju i lokalizaciji.

Slika 3: Slamtec RPLidar A2M8

Uobičajena frekvencija skeniranja iznosi 10 Hz što je ekvivalent 600 okretaja/min i uslijed ovih uvjeta razlučivost će biti 0.9°. Stvarna se frekvencija može prilagoditi u rasponu od 5 do 15 Hz prema zahtjevima korisnika.

Sam LiDAR skener sastoji se od jezgre skenera i mehaničkog napajanja – dijela zbog kojeg se jezgra skenera okreće velikom brzinom i u normalnom se načinu rada okreće i skenira u smjeru kazaljke na satu. Korisnicima je omogućeno dohvatiti podatke o skeniranju putem

komunikacijskog sučelja LiDAR-a i kontrolirati pokretanje, zaustavljanje i brzinu rotacije motora preko PWM[5]-a.

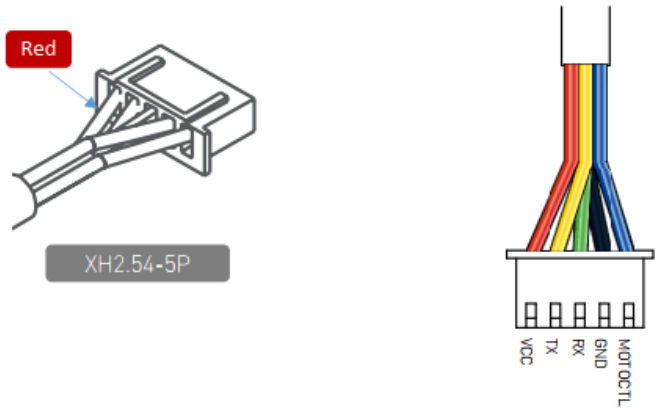
Tijekom svakog skeniranja i određivanja udaljenosti RPLidar emitira modulirani infracrveni laserski signal koji se tada odbija od ciljne površine objekta koji je potrebno otkriti. Povratkom signala on se detektira sustavom za prikupljanje informacija koji je ugrađen u sam uređaj te započinje obradu primljenih vrijednosti (kvaliteta povratnog signala, kut, udaljenost).

Uređaj koristi infracrveni laser male snage kao svjetlosni izvor i pokreće ga moduliranim impulsom. S obzirom da laser emitira svjetlost u vrlo kratkom vremenskom periodu, siguran je za ljude i kućne ljubimce, a doseže klasu 1 (Class 1[6]) laserskih sigurnosnih standarda. [5]

Uređaj se može koristiti u sljedećim primjenama:

- Opća navigacija i lokalizacija robota
- Skeniranje okoliša i 3D modeliranje
- Industrijski robot koji radi dulje vrijeme
- Kućni roboti (usisavanje, košnja trave i sl.)
- Mapiranje

Za napajanje uređaja je potrebno 5V istosmjerne struje . Standardni RPLidar A2 koristi muški priključak XH2.54-5P koji je prikazan u Slika 4:



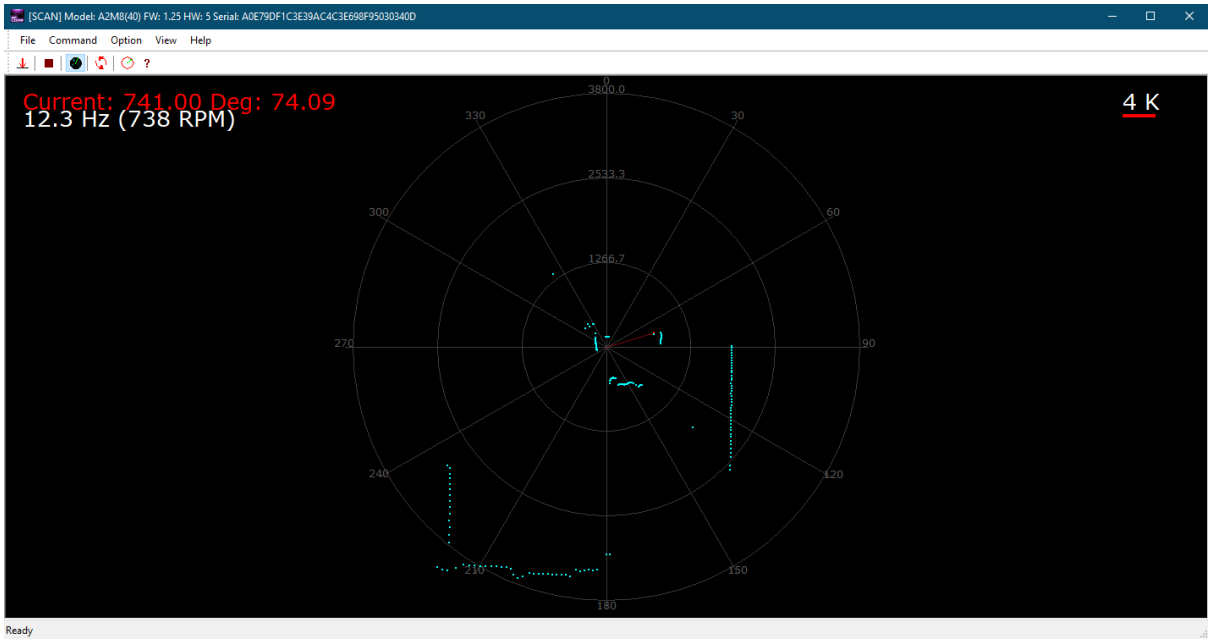
Slika 4: Priključak napajanja i komunikacije

[5] Pulsno širinska modulacija (eng. Pusle-width modulation)
[6] Laser je siguran tijekom svih načina normalnog rada ukoliko se njim rukuje propisanim uputama

Moguće je spojiti RPLidar na računalo preko USB kabla putem specifičnog konektora (Slika 5) koji olakšava upravljanje samim uređajem, dok se priključak prikazan na gornjoj slici (XH2.54-5P) također povezuje na konektor.

Slika 5: USB konektor

Tvrtka Slamtec također je osigurala grafičko sučelje (polarni dijagram) za pregled rezultata koje pruža LiDAR kao i SDK[7] da se ubrza razvoj programa krajnjeg korisnika.



Slika 6:Grafičko sučelje (polarni dijagram) tvrtke Slamtec

[7] Eng. Software development kit

4. RASPBERRY PI

Raspberry Pi pristupačno je računalo veličine kreditne kartice (mikro-računalo) koje ima mogućnost spajanja na računalni monitor ili TV uređaj i može koristiti standardnu USB periferiju poput miša i tipkovnice. Upravo to malo računalo omogućava ljudima svih dobnih skupina da istražuju računarstvo i nauče programirati u jezicima poput Python, Scratch i C++. Računalno je sposobno raditi sve što bi i „obično“ stolno računalo moglo raditi, od pregledavanja interneta i reprodukcije videozapisa visoke rezolucije, do izrade proračunskih tablica, obrade teksta i igranje nezahtjevnih igara.

Slika 7: Raspberry Pi 4 Model B

Neki ljudi koriste Raspberry Pi kako bi naučili programirati, dok ga ljudi s usvojenim znanjem programiranja koriste za programiranje vlastite elektronike za određene fizičke projekte. Štoviše, Raspberry Pi ima mogućnost interakcije s vanjskim svijetom i korišten je u širokom spektru projekata digitalnih entuzijasta, od glazbenih strojeva do meteoroloških postaja i cvrkutajućih kućica za ptice s infracrvenim kamerama. [6]

Već postoji nekoliko generacija Raspberry Pi računala (od 0 što je najstarije do 4 što je najnovije) i od svakog po 2 modela, model A i model B. Razlika između ova dva modela je da Model B ima više IO priključaka što model A klasificira kao skromniju verziju.

Raspberry Pi Foundation razvio je prilagođen operativni sustav s grafičkim sučeljem koji se temelji na Linux Ubuntu OS-u i naziva ga Raspberry Pi OS. Prethodne verzije Raspberry Pi OS-a bile su samo 32-bitne verzije i temeljile su se na Debianu koristeći ime Raspbian. Budući da novije 64-bitne verzije više ne koriste Debian nego Ubuntu, naziv je promijenjen u Raspberry Pi OS za 64-bitnu i 32-bitnu verziju. Od 1. veljače 2021. 64-bitna verzija je u beta verziji i nije prikladna za opću upotrebu. [7]

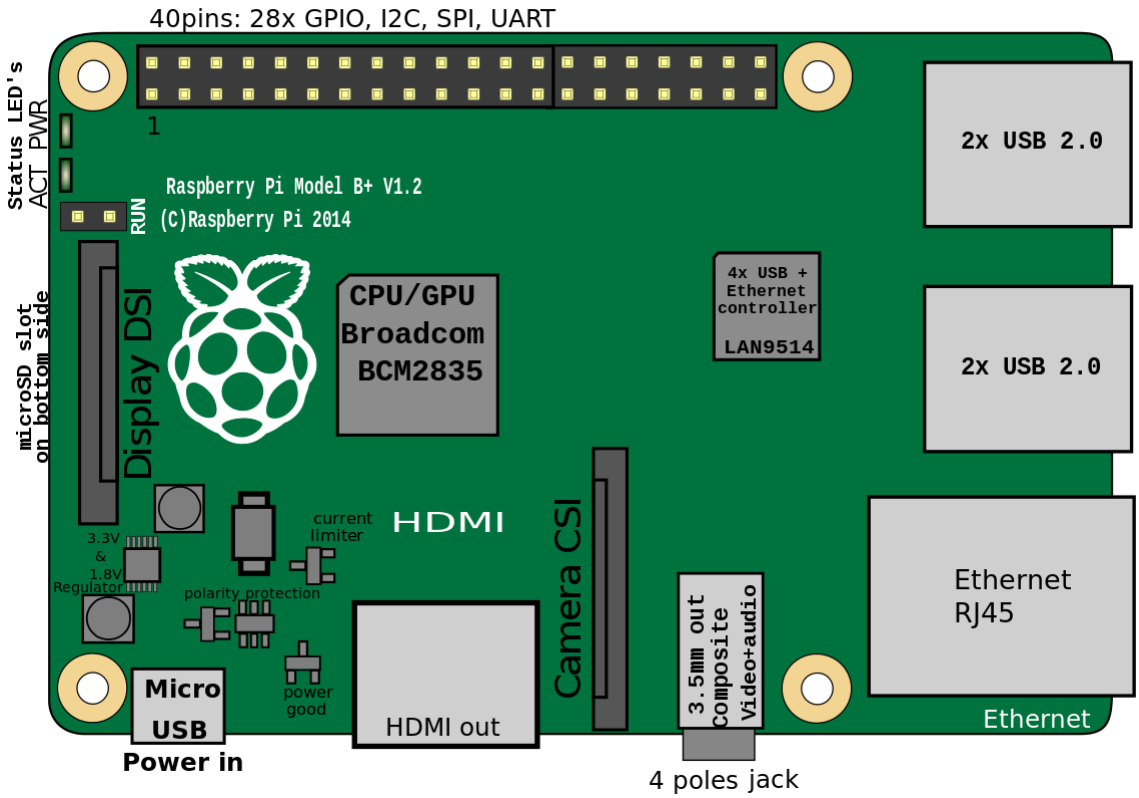
U ovom će se završnom radu koristiti Raspberry Pi 2 s 32-bitnom verzijom Raspberry Pi OS.

4.1. Raspberry Pi 2 Model B

Raspberry Pi 2 druga je generacija Raspberry Pi računala koja je u veljači 2015. godine zamijenila prvu generaciju navedenih mikro-računala. Iako su predstavljene i novije generacije Raspberry Pi 2, ostat će u proizvodnji barem do siječnja 2026. godine. Računalo sadrži četvero-jezgreni ARM Cortex-7 procesorsku jedinicu s radnim taktom od 900 MHz, 1 GB RAM memorije, 4 USB priključka, 40 GPIO[8] pinova (Slika 6), HDMI priključak, sučelje za ekran i kameru, Micro SD utor za karticu i VideoCore IV 3D grafički čip.

Računalo se može napajati preko vanjskog napajanja 5V i minimalno 2A ili preko USB priključka na računalu. Komunikacija s uređajem priključenim na osobno računalo vršit će se putem SSH[9] protokola, a veza s mikro-računalom biti će uspostavljena ethernet vezom s obzirom da Raspberry Pi 2 računalo nema mogućnost bežičnog spajanja na internet (WLAN). Za lakše upravljanje mikro-računalom uspostavit će se upravljanje preko udaljenog zaslona[10] na osobnom računalu - što će nam omogućiti isto iskustvo rada na Raspberry Pi računalu kakvo bi bilo da se mikroračunalo direktno spoji na monitor putem aplikacije Microsoft Remote Desktop. Za uspostavljanje veze također je potrebno na Raspberry Pi računalu instalirati xrdp (besplatna i open-source implementacija Microsoft Remote Desktop Protokola) koji omogućuje sustavima različitim od Microsoft Windows-a da uspostave vezu preko udaljenog zaslona.

8 General-purpose input/output
9 Secure shell protokol
10 Eng. Remote desktop

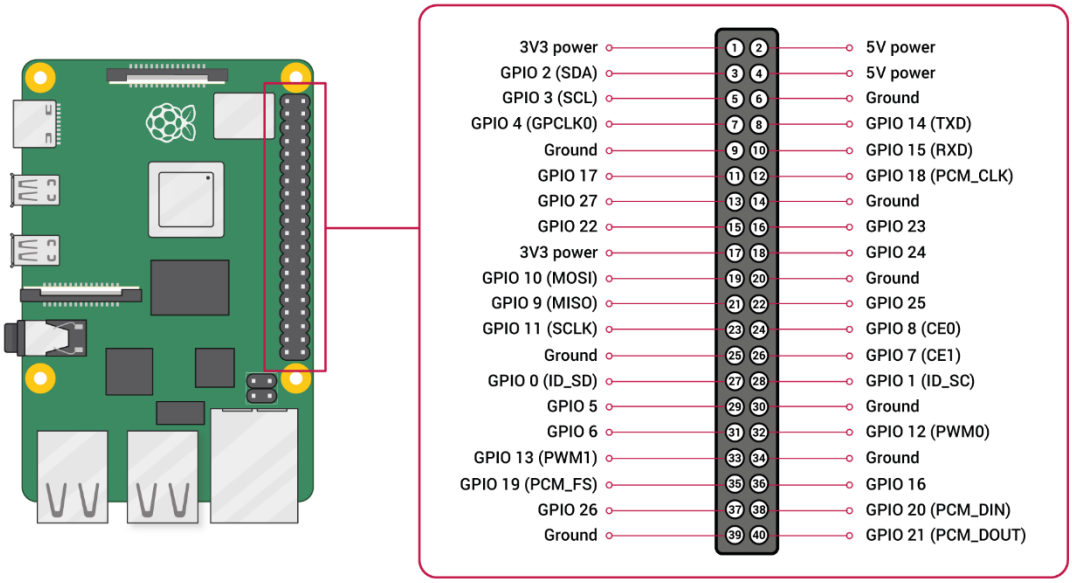


Slika 8: Prikaz konektora i glavnih integriranih krugova Raspberry Pi 2 računala

Komunikacija vanjskog hardvera (raznorazni senzori) s računalom vrši se preko pinova. Postoje dva 5V i dva 3.3V pina za napajanje senzora i nekoliko „nula“ za uzemljenje dok su preostali pinovi kompatibilni s 3.3V, odnosno mogu predati 3.3V ili primiti 3.3V. Izlaz output pinova može biti visok (3.3V) ili nizak (0V) što vrijedi i za input pinove. Osim jednostavnih input i output pinova, mogu se koristiti i oni s raznim alternativnim funkcijama gdje su neke dostupne na svim pinovima, dok su neke samo na određenima [8]:

- PWM
    - Softverski dostupno na svim pinovima, a hardverski na GPIO12, GPIO13, GPIO18 i GPIO19

- I2C[11]
  - Dostupno na pinovima GPIO0, GPIO1, GPIO2 i GPIO3
- SPI[12]
  - Dostupno na pinovima GPIO7, GPIO8, GPIO9, GPIO10, GPIO11, GPIO16, GPIO17, GPIO18, GPIO19, GPIO20 i GPIO21
- Serial
  - Dostupno na pinovima GPIO14 i GPIO15



Slika 9: GPIO pinovi na Raspberry Pi 2 uređaju

[11] Eng. Inter-Integrated Circuit
[12] Eng, Serial Peripheral Interface

5. PYTHON

Python programski je jezik visoke razine i opće namjene. Pythonovoj filozofiji dizajna na prvom je mjestu čitljivost koda uporabom značajnog razmaka. Njegov objektno orijentirani pristup pomoć je programerima da napišu jasan i logičan kod za male, ali isto tako i za velike projekte. Podržava više paradigmi programiranja, uključujući strukturirano (posebno proceduralno), objektno orijentirano i funkcionalno programiranje. [9]

Razvijen je krajem 1980-ih, a 1991. godine objavio ga je Guido van Rossum kao nasljednika programskog jezika ABC. Dobio je ime prema komičnoj seriji *Monty Python's Flying Circus*, iz 1970-ih godina. Python 2.0, objavljen 2000. godine predstavio je nove značajke. Python 3.0, objavljen 2008. godine, bio je glavna revizija jezika koji nije u potpunosti unatrag kompatibilan, a velik dio Python 2 koda ne radi neizmijenjen na Pythonu 3. S krajem Pythona 2, podržani su samo Python 3.6.x i novije verzije, a starije verzije i dalje podržavaju npr. Windows 7 (i stariji operacijski sustavi koji nisu ograničeni na 64-bitni Windows). [9][10]

Od siječnja 2021. Python zauzima treće mjesto u TIOBE-ovom indeksu najpopularnijih programskih jezika, iza C-a i Jave, prethodno osvojivši drugo mjesto i nagradu za najveći rast popularnosti u 2020. godinu.

Temeljna filozofija jezika sažeta je u dokumentu *Zen of Python* (PEP 20), koji uključuje aforizme poput [11]:

- Bolje je lijepo nego ružno
- Bolje je eksplicitno nego implicitno
- Bolje je jednostavno nego složeno
- Bolje je kompleksno nego komplicirano
- Čitljivost se također računa

Navedena filozofija olakšava programiranje u Pythonu zbog lakog učenja sintakse, dok početnicima pruža ugodno i jednostavno iskustvo učenja novog programskog jezika i programiranja.

Umjesto da svu svoju funkcionalnost ugradi u jezgru programa, Python je dizajniran da bude vrlo proširiv. Upravo ta kompaktna modularnost učinila ga je posebno popularnim kao način

dodavanja programabilnih sučelja postojećim aplikacijama. Smišljen je kao lako čitljivi jezik. Za razliku od mnogih drugih jezika, on ne koristi vitičaste zagrade za razgraničenje blokova, a točka zarez nakon izraza nisu obavezni i ima manje sintaktičkih iznimaka i posebnih slova kao C ili Pascal. [12]

Primjer Python koda:

```
lidar = RPLidar(None, '/dev/ttyUSB0')
lidar.set_pwm(660)
time.sleep(2)


def lidar_scans(self, max_buf_meas=500, min_len=360):

        lidar.stop()
        scan = []
        iterator = lidar.iter_measurments(max_buf_meas)
        for new_scan, quality, angle, distance in iterator:
            if new_scan:
                if len(scan) > min_len:
                    yield scan
                scan = []
            if quality > 0 and distance > 0:
                scan.append((quality, angle, distance))



for scan in lidar_scans(lidar):
    if ekran.provjeri_exit():
        lidar.set_pwm(0)
        lidar.stop()
        lidar.disconnect()
        ekran.exit()

    lista_tocaka = lista_podataka_u_listu_tocaka(scan)
    lista_prepreka = grupiraj_prepreke(lista_tocaka, 300)
    ekran.update(lista_prepreka)
```

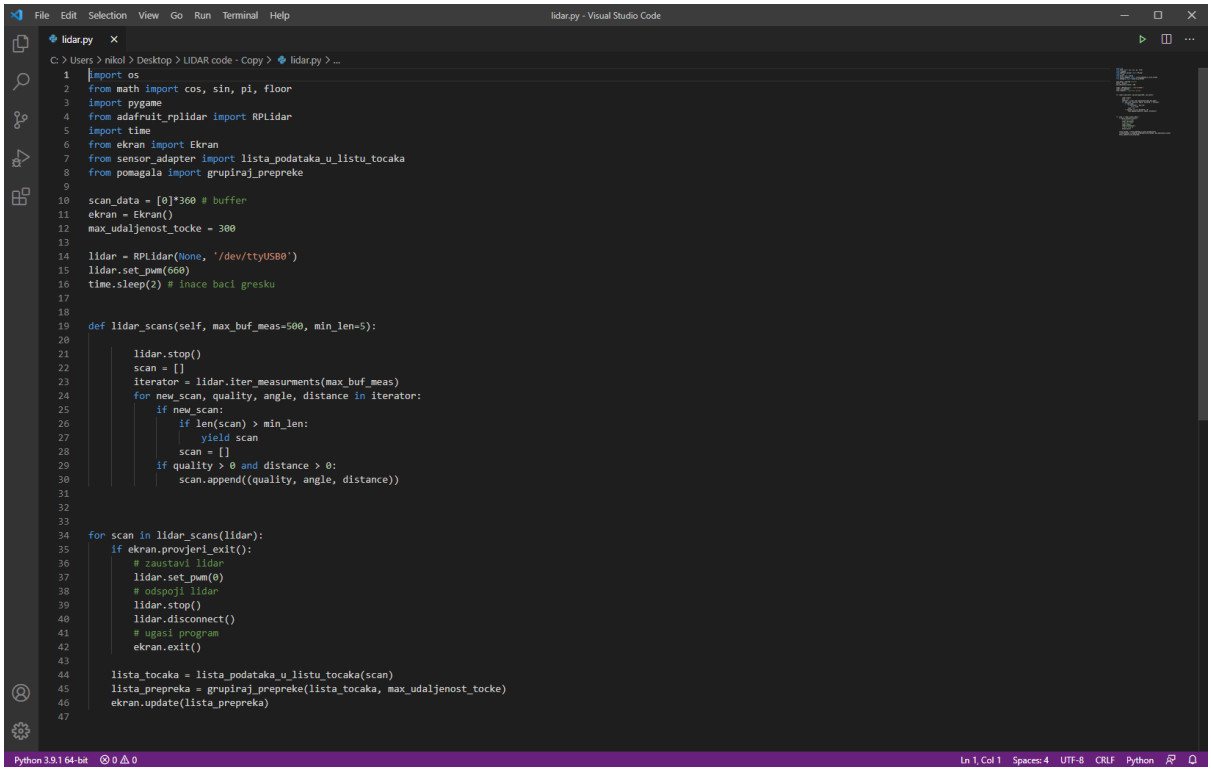
Pythonom se može pisati u mnogim alatima, no u ovom će se završnom radu koristiti programski paket Microsoft Visual Studio Code.

5.1. Microsoft Visual Studio Code

Programski paket Microsoft Visual Studio Code (kratko VS Code) vrlo je moćan besplatan open-source editor tvrtke Microsoft za Windows, Linux i MacOS. VS Code prvi je put objavljen 29. travnja 2015. godine, dok je pod licencom MIT-a objavljen 18. studenog 2015.

Korisnicima omogućuje otvaranje jednog ili više direktorija koji se potom mogu spremiti u radne prostore za buduće korištenje. Podržava brojne programske jezike poput Java, JavaScript, Node.js, C++, Python i slično dok se skup značajki razlikuje po jezicima. Dostupna su mnoga proširenja (tzv. extensions) putem centralnog repozitorija što uključuje podršku jezika i dodatke editoru. Uključuje i više proširenja za FTP (File Transfer Protocol) što ga čini besplatnom alternativom za web razvoj zbog toga što se kod može sinkronizirati između editora i poslužitelja bez preuzimanja dodatnih programskih paketa. [13]

Zbog značajki poput podrške za otklanjanje pogrešaka (debugging), formatiranje sintakse, inteligentnog dovršavanja koda, refaktoriranja koda, ugrađenog Git-a, mogućnosti mijenjanja teme, stvaranje prečaca i mnogih drugih, danas se nalazi na prvom mjestu korištenih kod editora s udjelom od 50.7% prema istraživanju tvrtke Stack Overflow. [14]



Slika 10: Grafičko sučelje programskog paketa VS Code

6. RAZRADA PROBLEMA

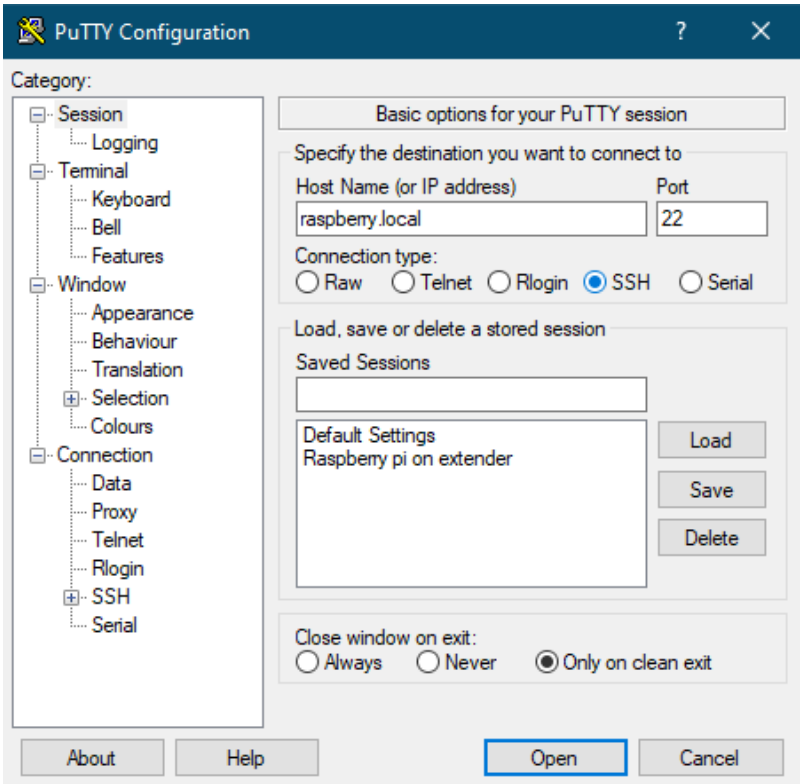
Potrebna oprema korištena u ovom radu je RPLidar A2M8, Raspberry Pi 2 mikro-računalo, Micro SD kartica, UTP kabel, USB-A na Micro-USB kabel i osobno računalo.

6.1. Priprema Raspberry Pi računala

Najprije je potrebno instalirati Raspberry Pi OS na Micro SD karticu i staviti ju u Raspberry Pi. Nakon što je to učinjeno, mikro-računalo se priključuje na napajanje i na osobno računalo pomoću UTP kabla. Nakon pokretanja osobnog i mikro-računala potrebno je uspostaviti serijsku vezu između njih što će biti urađeno putem programskog paketa PuTTY.

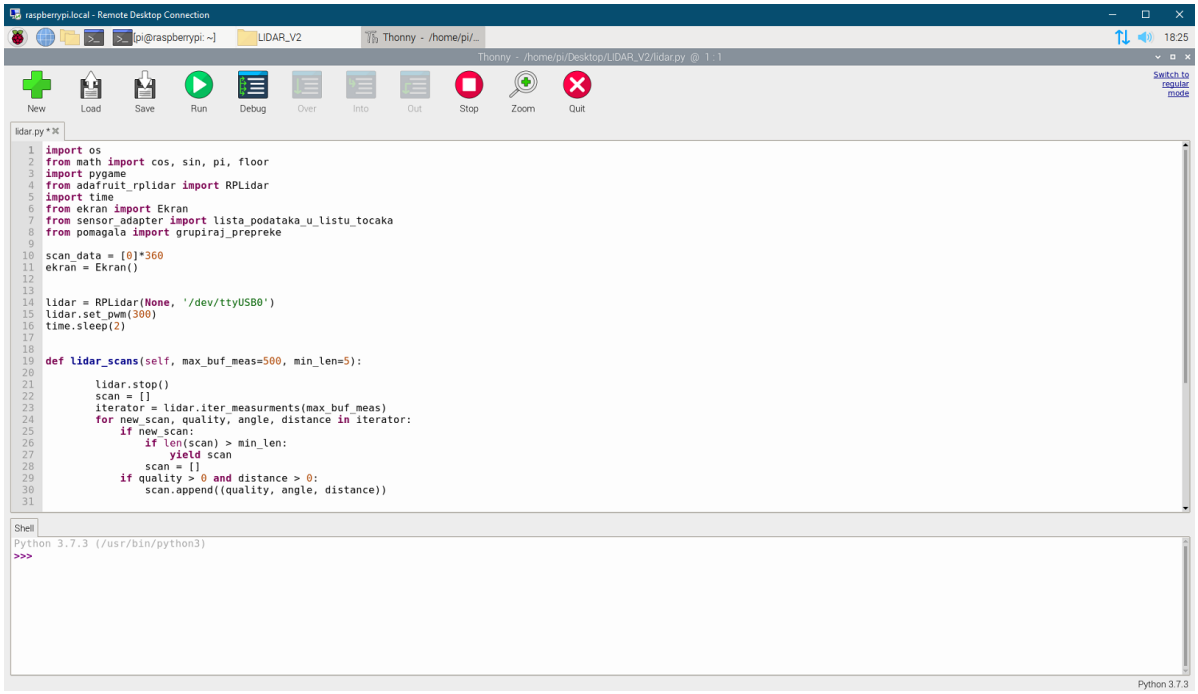
PuTTY besplatni je open-source terminal emulator, serijska konzola i aplikacija za prijenos datoteka. Podržava nekoliko mrežnih protokola, uključujući SCP, SSH, Telnet, rlogin i raw socket vezu, a sam naziv PuTTY nema posebno značenje. Grafičko sučelje programa vidljivo je na slici 11.

Slika 11.



Slika 11: Grafičko sučelje programskog paketa PuTTY

Nakon što je, pomoću programskog paketa PuTTY , pristupljeno Raspberry Pi računalu, instaliran je xrdp kako bi se omogućio pristup preko udaljenog zaslona što se ostvaruje pomoću sljedeće komande u PuTTY-ju: *sudo apt install tightvncserver* i *sudo apt install xrdp.* Pokretanjem aplikacije Microsoft Remote Desktop, osobnim se računalom spaja na Raspberry Pi pomoću udaljenog zaslona. Pokretanjem programskog paketa Thonny na Raspberry Pi-ju može se otvoriti Python kod koji pokreće LiDAR skener, prikuplja i obrađuje informacije koje prikazuje u polarnom dijagramu.



Slika 12: Grafičko sučelje programskog paketa Thonny preko udaljenog zaslona

6.2. Prikupljanje podataka

Da bi se olakšalo rješavanje zadatka, korištena je postojeća biblioteka *adafruit_rplidar.py* za upravljanje LiDAR skenerom. Ova datoteka proizvod je tvrtke Adafruit Industries, licencirana je od strane MIT[13]-a što ju na prvi pogled čini pouzdanom. Nakon vrlo detaljnog proučavanja datoteke i usporedbe njezinog koda s dokumentacijom i komunikacijskim protokolom RPLidar A2M8 uređaja dolazi se do zaključka da je biblioteka *adafruit_rplidar.py* ispravna za korištenje s navedenim uređajem i da će prema odgovarajućim očekivanjima raditi.

[13] Massachusetts Institute of Technology

Uz datoteku *adafruit_rplidar.py* priložena je i datoteka *sample.py* koja može poslužiti kao pomoć pri pokretanju i prikupljanju podataka s LiDAR skenera.

Nakon instaliranja Python 3.9.1 i programskog paketa Microsoft Visual Studio Code pokrenut je navedeni programski kod, no podaci s LiDAR skenera nisu prikupljeni zbog toga što se ni sam LiDAR skener nije pokrenuo. To dovodi u pitanje pouzdanost navedenih Python kodova. Detaljnijim uspoređivanjem *adafruit_rplidar.py* datoteke i dokumentacije priložene s uređajem zaključeno je da je ispravna, no još se uvijek nameće pitanje „Zašto ne radi?“ stoga je zatražena pomoć mentora prof. dr. sc. Mladena Crnekovića, dipl. ing., koji napominje da osobno računalo ne koristi serijsku komunikaciju preko USB priključka na jednak način kao što to Raspberry Pi računalo radi. Pokretanjem programskog koda na mikro-računalu, program je proradio. Odlučeno je da se programski kod piše i razvija na osobnom računalu u programskom paketu Microsoft Visual Studio Code zbog velikog zaostajanja udaljenog zaslona i zbog bržeg rada računala, no testiranje koda mora se vršiti na Raspberry Pi računalu iz prethodno navedenog razloga.

Daljnjim proučavanjem programskog koda *sample.py* pronađen je dio koji prikuplja podatke s LiDAR skenera i oko kojeg će biti bazirano kompletno rješenje ovog završnog rada:

```
def lidar_scans(self, max_buf_meas=500, min_len=5):

        lidar.stop()
        scan = []
        iterator = lidar.iter_measurments(max_buf_meas)
        for new_scan, quality, angle, distance in iterator:
            if new_scan:
                if len(scan) > min_len:
                    yield scan
                scan = []
            if quality > 0 and distance > 0:
                scan.append((quality, angle, distance))
```

gdje je vidljivo da se podaci oblika (kvaliteta signala, kut, udaljenost) spremaju u polje *scan.*

6.3. Pretvaranje polarnih koordinata u kartezijeve

Prikupljanje podataka s LiDAR skenera zahtijevalo je njihov prikaz u polarnom dijagramu, no prije nego je to moguće, potrebno je pretvoriti polarne koordinate u kartezijeve koordinate da nam se omogući crtanje točaka na zaslonu. Svaka se točka opisuje kvalitetom (zanemaruje se), kutom i udaljenošću i vrlo lako se pretvara u kartezijeve koordinate iz polarnih sljedećim algoritmom:

- Potrebno je kut iz stupnjeva pretvoriti u radijane:

$$kut_s = \text{kut u stupnjevima} \tag{2}$$

$$kut_r = \text{kut u radijanima} \tag{3}$$

$$kut_r = \frac{kut_s}{180°} \cdot \pi \tag{4}$$

- Potrebno je naći udaljenost po *x*-osi:

$$x = udaljenost \cdot \cos(kut_r) \tag{5}$$

- Potrebno je naći udaljenost po *y*-osi:

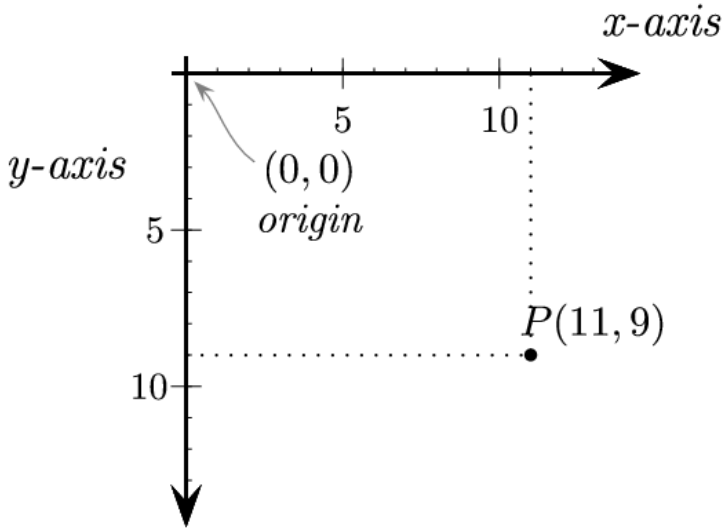
$$y = udaljenost \cdot \sin(kut_r) \tag{6}$$

Nakon što su vrijednosti *x* i *y* definirane, potrebno je odrediti njihov položaj u ekranu širine *width*, visine *height* i maksimalne udaljenosti točke *max_distance* u kojemu će biti prikazan polarni dijagram gdje je *max_distance* odgovoran za skaliranje veličine dijagrama, a ishodište kartezijevog (pravokutnog) koordinatnog sustava u gornjem je lijevom kutu gdje *x*-os gleda u desno, a *y*-os prema dolje kao što je prikazano u Slika 13:

$$max_d = max\_distance \tag{7}$$

$$položaj = \left(\frac{width}{2} + \frac{x}{max_d} \cdot \frac{width}{2}, \quad \frac{height}{2} + \frac{y}{max_d} \cdot \frac{height}{2}\right) \tag{8}$$

gdje je *položaj* točka u ravnini ekrana na kojem se iscrtava dijagram.



Slika 13: Primjer navedenog koordinatnog sustava

6.4. Udaljenost dviju točaka, prepreka i prolaz

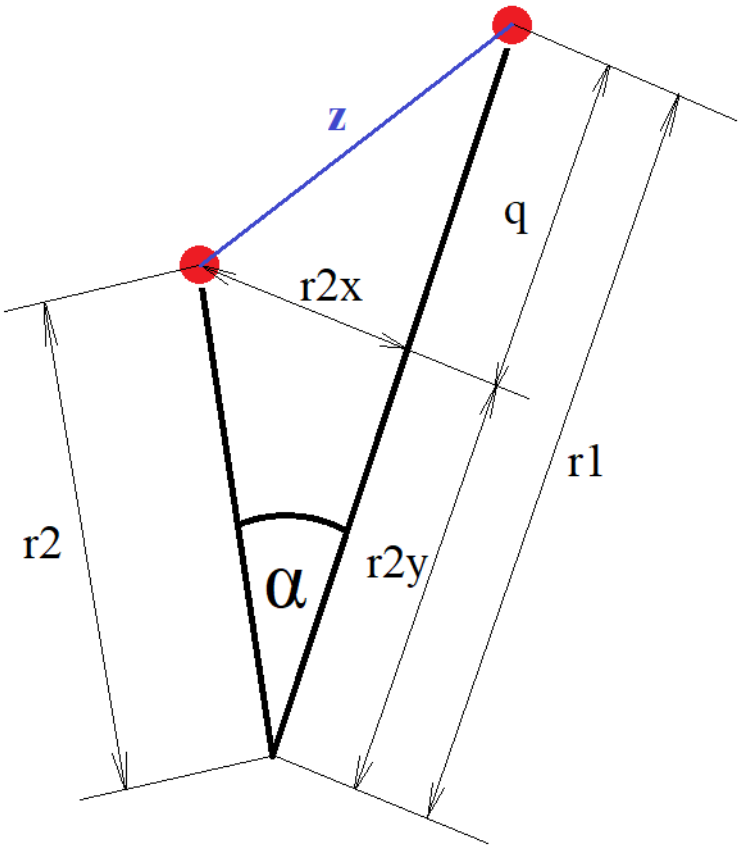
S obzirom da je prikupljanje podataka temelj za daljnje rješavanje završnog zadatka, nakon toga potrebno je u programu definirati pojam prepreke. Prepreka je skup točaka u kojoj susjedne točke nisu udaljene više od *max_udaljenost_tocke*, tj. ukoliko je udaljenost dviju točaka manja od *max_udaljenost_tocke* smatrat će se da je to jedna cjelina u koju se dodaju sve točke koje su od zadnje dodane točke udaljene manje od *max_udaljenost_tocke*. Pomoć za objašnjenje algoritma udaljenosti između dviju točaka prikazan je u Slika 14 gdje je *r1* udaljenost točke 1, *r2* udaljenost točke 2, α je kut između točke 1 i 2, z je udaljenost između dvije točke, r2y je projekcija vektora r2 na vektor r1, r2x je projekcija vektora r2 u normalnom smjeru vektora r1, dok su relacije su prikazane jednadžbama (9), (10), (11) i (12):

$$r_{2x} = r_2 \cdot \sin(\alpha) \tag{9}$$

$$r_{2y} = r_2 \cdot \cos(\alpha) \tag{10}$$

$$q = r_1 - r_{2y} \tag{11}$$

$$z = \sqrt{q^2 + r_{2x}^2} \tag{12}$$



Slika 14: Skica algoritma za udaljenost dviju točaka

Računanje udaljenosti između dviju točaka i definiranje prepreka dovodi do definiranja prolaza. Prolaz je određen kao prostor između dviju prepreka koji je veći od 500 mm gdje se promatraju krajnja točka neke prepreke i početna točka sljedeće. Ukoliko je prolaz veći od maksimalne veličine prolaza neće se smatrati prolazom zbog toga što dolazi do mogućnosti nedostatka podataka o tom prostoru jer se zraka reflektira od sjajne površine. Ukoliko su mu dimenzije između 500 mm i maksimalne veličine prolaza, prolaz će biti definirani kao zeleni s obzirom da je minimalna potrebna širina za prolaz mobilnog robota definirana na 500 mm.

6.5. Implementiranje algoritama u programski kod

S obzirom da kompletan programski kod ovisi o podacima koje LiDAR skener predaje, na priloženom programskom kodu u poglavlju **6.2 Prikupljanje podataka** oko kojeg će se graditi program, temeljiti će se rješenje ovog rada. Za lakše snalaženje i pisanje programa koristit će se načela objektno orijentiranog programiranja.

*** Svi programski kodovi priloženi su na kraju dokumenta. ***

6.5.1. Definiranje točke i liste točaka

Algoritam za prikupljanje podataka s LiDAR skenera kao rezultat daje listu običnih podataka koju je potrebno pretvoriti u listu točaka.

Tocka je klasa u datoteci *tocka.py* koja sadrži dva podatka: kut i udaljenost, dok se kvaliteta zanemaruje. Podaci se pretvaraju u tip *Tocka* tako da svakom podatku sa senzora definiramo vrijednosti kuta i udaljenosti. Dodavanjem više točaka u listu dobije se lista u kojoj su svi elementi tipa *Tocka*. Da bi program bio univerzalan i kompatibilan s drugim uređajima, potrebno je definirati *sensor_adapter.py* datoteku (izmjenjuje se s obzirom na izlazne podatke različitih uređaja) koja pretvara podatak s uređaja u *Tocku*:

```
def podatak_senzora_u_tocku(sensor_data: (float, float, float)) -> Tocka:
    (_, angle, distance) = sensor_data
    return Tocka(angle, distance)
```

gdje je nakon tog postupka *Tocka* ubačena u listu točaka.

6.5.2. Definiranje prepreke i udaljenosti dviju točaka

Nakon što je definirana klasa *Tocka* i lista podataka konvertirana u listu točaka, potrebno je definirati i odrediti prepreke iz zadane liste točaka. Za pojam prepreke također je uvedena klasa *Prepreka* s tri podatka: lista točaka u prepreci, duljina i najbliža točka.

Prepreka je definirana kao skup točaka u kojoj svaka sljedeća točka od prethodne nije udaljenija od *max_udaljenost_tocke* - što je u ovom slučaju jednako vrijednosti 300 mm. Ako je taj uvjet zadovoljen, *Tocka* se pridružuje u postojeću listu točaka u prepreci i prepreka se povećava za 1 točku. U suprotnom, ako uvjet nije zadovoljen, određena prepreka se pridružuje u listu prepreka u kojoj su svi elementi tipa *Prepreka*.

Duljina prepreke računa se kao zbroj udaljenosti svih susjednih točaka u prepreci, dok je najbliža točka prepreke definirana kao točka u listi točaka prepreke koja ima najmanju vrijednost *distance.*

Za računanje udaljenosti dviju točaka također je stvorena pomoćna datoteka *udaljenost_tocke.py* u kojoj je definirana funkcija za računanje udaljenosti dviju točaka kojoj su ulazni argumenti dvije *Tocke* i čiji je algoritam opisan u poglavlju **6.4 Udaljenost dviju točaka, prepreka i prolaz**.

6.5.3. Definiranje prolaza

Definiranje prepreka indirektno daje definiciju prolaza – prostor između dviju prepreka. Definirana je klasa *Prolaz* koja sadrži 3 podatka: početna točka, krajnja točka i veličina, tj. duljina. Početna točka prolaza zadnja je točka prepreke, krajnja točka prolaza prva je točka sljedeće prepreke, a veličina udaljenost tih dviju točaka. Funkcijom *ekstrakcija_prolaza,* kojoj je argument lista prepreka, izlučujemo prolaze i pridružujemo ih u listu prolaza gdje je svaki element tipa *Prolaz.* S obzirom da for petlja ide od prve do zadnje prepreke, zanemaruje se prolaz od zadnje do prve te je potrebno provjeriti:

```
def ekstrakcija_prolaza(lista_prepreka: []) -> []:
    prolazi = []
    for i in range(len(lista_prepreka) - 1):
        prepreka1 = lista_prepreka[i]
        prepreka2 = lista_prepreka[i + 1]
        prolaz = Prolaz(prepreka1, prepreka2)
        prolazi.append(prolaz)
    prolaz = Prolaz(lista_prepreka[len(lista_prepreka)-1], lista_prepreka[0])
    prolazi.append(prolaz)
    return prolazi
```

6.5.4. Filtriranje anomalija

Kao i kod svakog drugog senzora tako i kod LiDAR skenera dolazi do grešaka u skeniranju koje će biti definirane kao anomalije. Moguća je pojava grešaka kao rezultat reflektiranja od sjajne površine, prevelike i premale udaljenosti predmeta ili jednostavno zbog nesavršenosti uređaja. Za lakše upravljanje s većim brojem vrijednosti na jednom kutu, potrebno je stvoriti rječnik u kojem su ključevi kut od 0° do 360°. Kako skener nije savršen, potrebno je kut zaokružiti na cjelobrojnu vrijednost (mogući scenarij: kut u prvom skeniranju iznosi 1.58°, a

drugom 1.63° i stvaraju se dvije vrijednosti za praktički isti kut). Greške su prikazane kao nasumične točke u polarnom dijagramu, a mobilni se robot neće gibati velikom brzinom, stoga je odlučeno da se uzima srednja vrijednost n brojeva - što je u programskom kodu definirano kao varijabla *broj_avg_podataka.*

Potrebno je stvoriti datoteku *anomalija_filter.py* u kojoj je definirana funkcija *dohvati_srednju_vrijednost_tocaka* kojoj je ulazni argument polje podataka koje daje LiDAR skener. Nakon primanja podataka, u rječnik se dodaje n broj udaljenosti točke koje su na istom kutu i iz njih se definira srednja vrijednost bez maksimalne i minimalne udaljenosti[14].

```
suma = sum(tocke) - min(tocke) - max(tocke)
broj_tocaka = len(tocke) - 2
avg_dist = suma / broj_tocaka
```

Nakon što se iscrta dijagram, u funkciju odlazi novo polje podataka, a staro izlazi (princip First In - First Out) i generira se nova srednja vrijednost.

6.5.5. Definiranje zaslona i crtanje

Nakon definiranja svih navedenih podataka, potrebno ih je iscrtati u polarnom dijagramu. Za tu svrhu potrebno je stvoriti datoteku *ekran.py* i klasu *Ekran* koja sadrži 4 podatka: širinu zaslona *width,* visinu zaslona *height*, maksimalnu udaljenost točke koja se prikazuje *max_distance* i boju ispune zaslona *background_boja*.

Izbacivanjem anomalija filtriranjem programskim kodom algoritma, opisanog u prethodnom poglavlju, dobije se filtrirana lista podataka koju je potrebno prebaciti u listu točaka - što je detaljnije opisano u poglavlju **6.5.1 Definiranje točke i liste točaka**. Nakon toga potrebno je prebaciti točke iz polarnih koordinata u kartezijeve. Prepreke će biti prikazane kao skup međusobno povezanih točaka, dok će prolazi biti prikazani kao crta između krajnje točke jedne i početne točke sljedeće prepreke. Koordinatne osi prikazat će se bijelom bojom. Kut raste u smjeru kazaljke na satu.

Potrebno je definirati boje prepreka: žutom je prikazana najbliža prepreka, ljubičastom najdulja prepreka, a preostale bijelom bojom. Prolazi veličine od *min_velicina_prolaza* (500 mm) do *max_velicina_prolaza* (1500 mm) prikazani su zelenom bojom, dok preostali neće biti prikazani (praznina). Na zaslonu je također potrebno ispisati informacije poput: najveće

[14] U statistici – eng. centered mean

prepreke, najbliže prepreke, najvećeg prolaza, najbližeg prolaza i broja prolaza – čiji će prikaz biti u gornjem desnom kutu.

Stvorena je još jedna pomoćna datoteka *pomagala.py* u kojoj su definirane funkcije za dohvaćanje najdulje prepreke i dohvaćanje najbliže prepreke. S obzirom da je u samoj prepreci definirana najbliža točka i udaljenost, ti će podaci biti referenca za definiranje navedenih podataka o prepreci. Kao i prepreka, prolaz u sebi sadrži veličinu pa je jednostavnim sortiranjem moguće izlučiti najveći prolaz, dok je za najbliži prolaz potrebno stvoriti funkciju koja definira udaljenost točke na sredini prolaza i nakon toga uspoređuje udaljenosti točaka na sredini svih prolaza.

Ispisivanjem svih informacija, prepreka i prolaza dobiju se rezultati programa.

7. REZULTATI

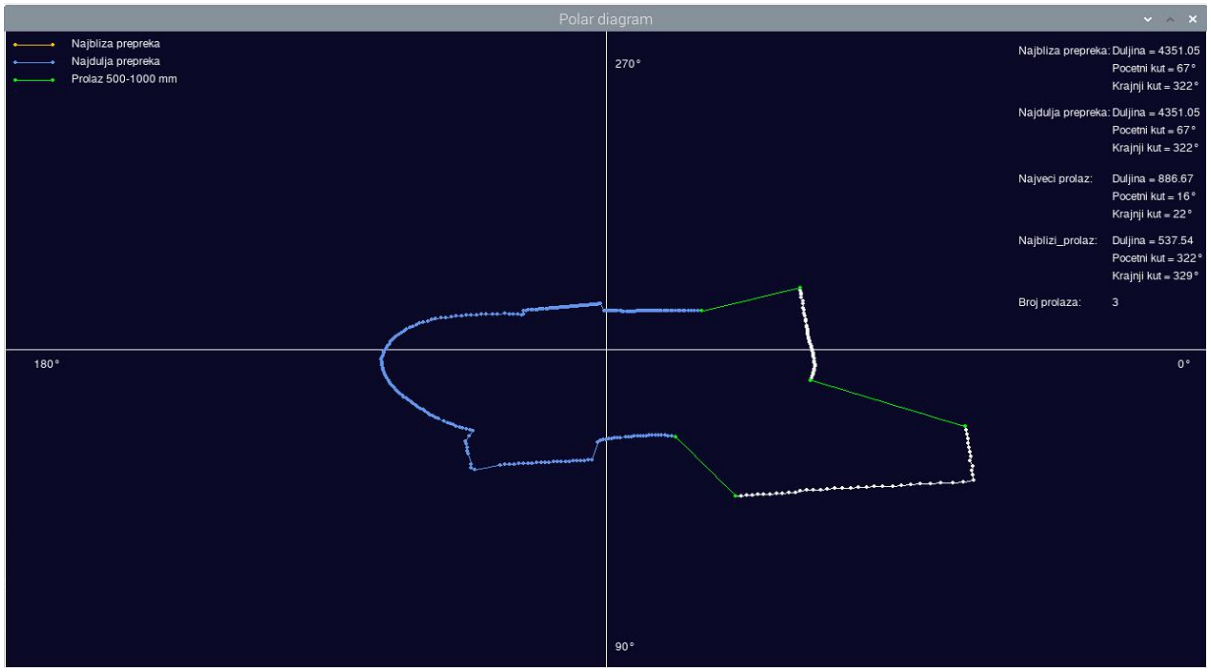
Postavljanjem poligona u željeni oblik, implementacijom programskog koda u programski paket Thonny na Raspberry Pi računalu i njegovim pokretanjem dobije se prikaz poligona u polarnom dijagramu i ispis podataka poput najveće i najbliže prepreke; najvećeg, najbližeg i broja prolaza.

Legenda boja za bolje razumijevanje polarnog dijagrama:

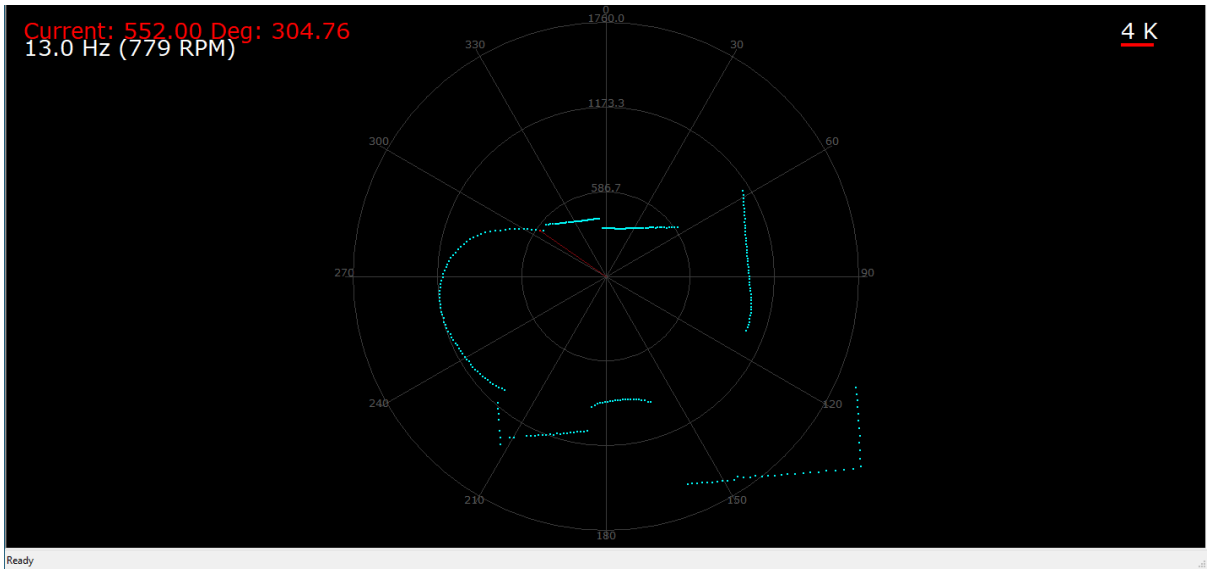
Slika 15: Legenda boja

7.1. Usporedba s tvorničkim prikazom

Slika 16: Poligon 1



Slika 17: Prikaz poligona 1 u polarnom dijagramu

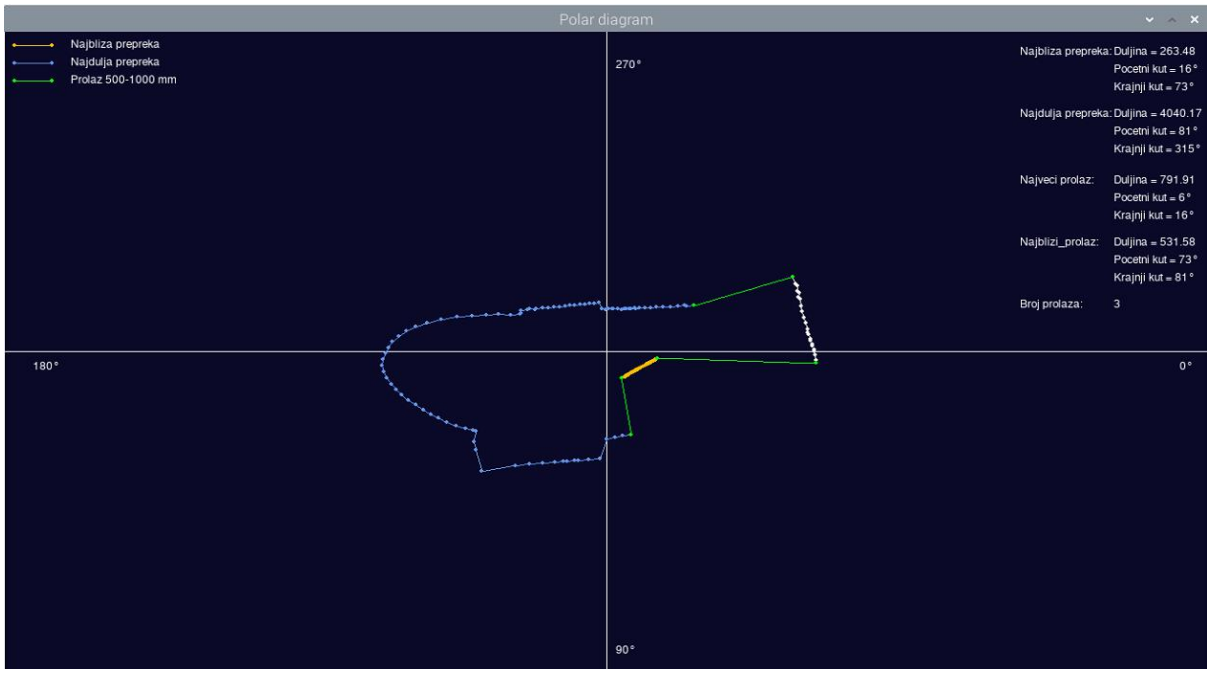


Slika 18: Prikaz poligona 1 u tvorničkom programu

Usporedbom realnog izgleda poligona 1, tvorničkog prikaza i prikaza iz programa priloženog na kraju dokumenta vidljivo je da je prikaz ispravan i vjerodostojan, no dolazi do razvlačenja slike u smjeru *x*-osi (zbog toga što se koordinate određuju u odnosu na širinu i visinu ekrana) što se vrlo lako može popraviti izjednačavanjem vrijednosti *width* i *height* koje definiraju širinu i visinu zaslona polarnog dijagrama.

7.2. Poligon 2

Slika 19: Poligon 2

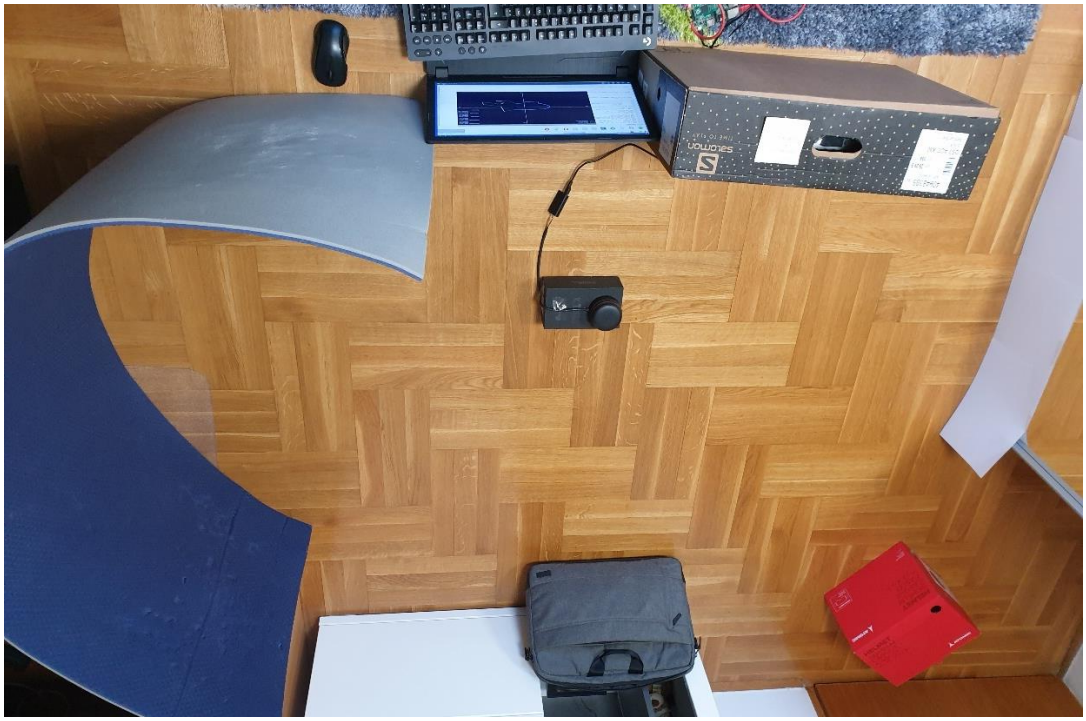


Slika 20: Polarni dijagram poligona 2

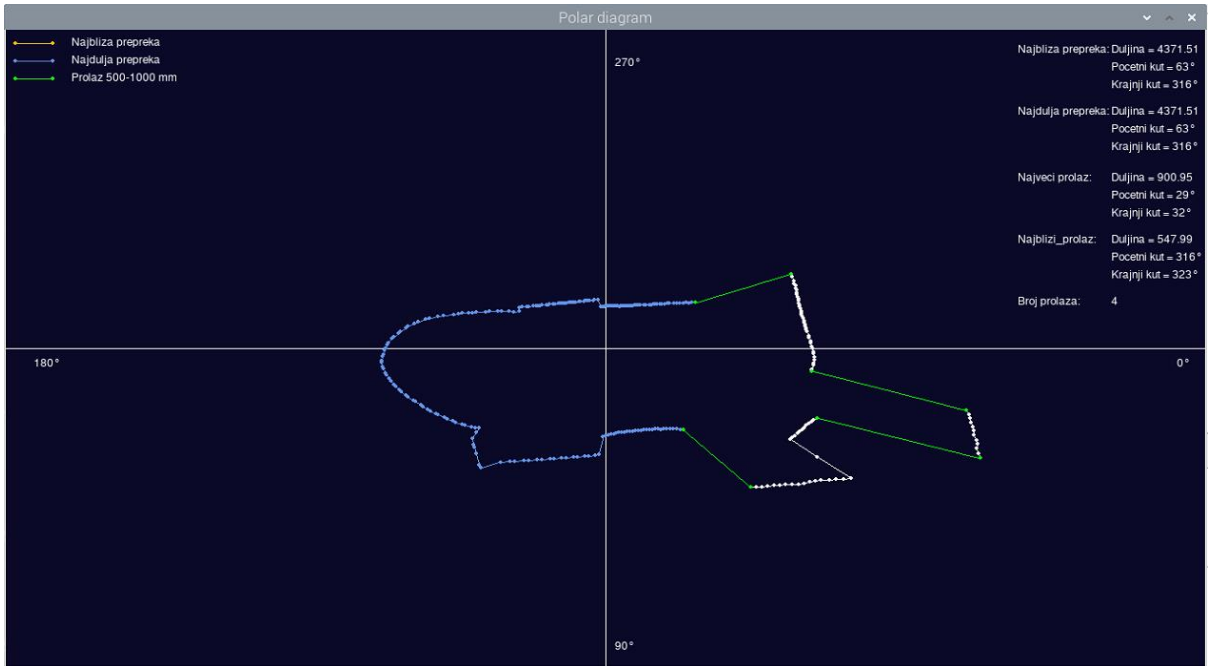
Slika 21: Ispis podataka za poligon 2

Prikazom poligona 2 u polarnom dijagramu i uz priložene informacije vidljivo je da je crvena kutija u Slika 19 najbliža prepreka i prikazana je žutom bojom, dok je najduža prepreka - skup prostirke, sive kutije i računala, prikazana ljubičastom bojom. Također, iz polarnog se dijagrama mogu očitati mogući prolazi koji su prikazani zelenom bojom i brojanjem se dolazi do zaključka da im je broj jednak broju u ispisu podataka.

7.3. Poligon 3



Slika 22: Poligon 3

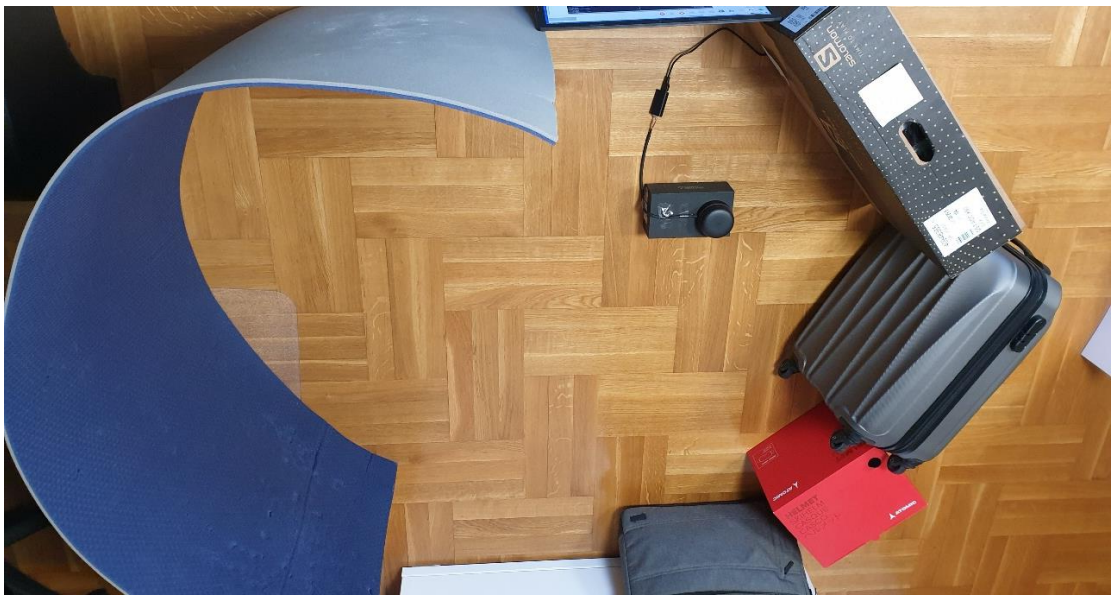


Slika 23: Polarni dijagram poligona 3

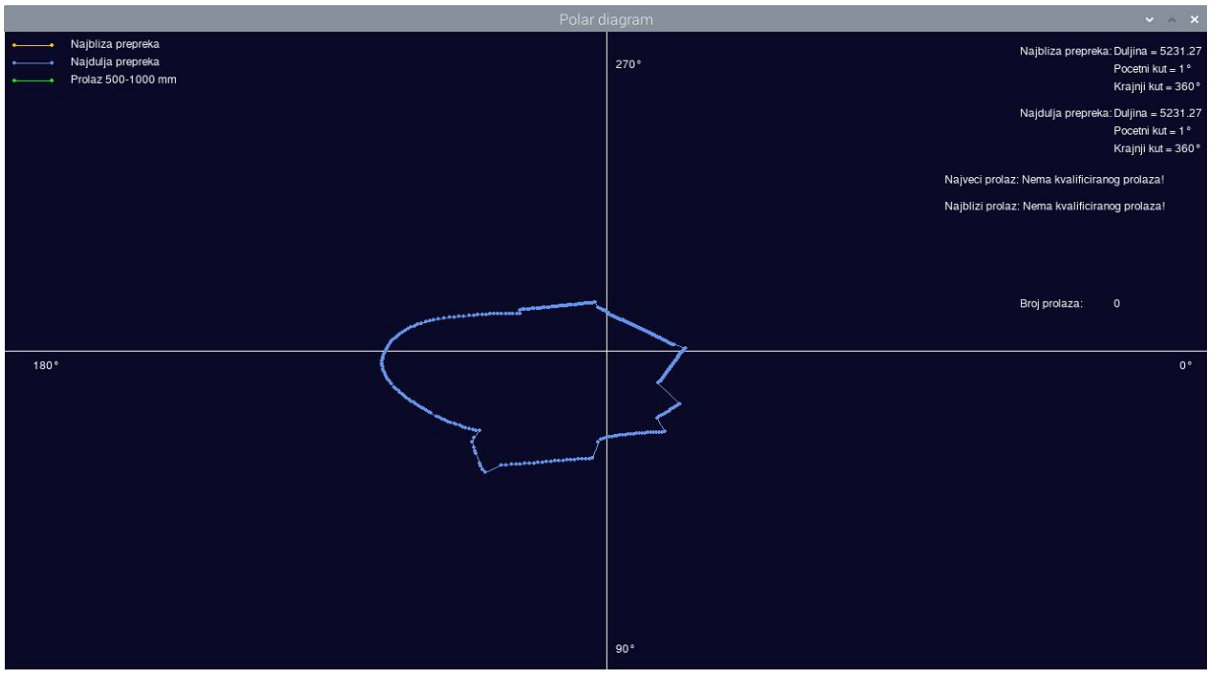
Slika 24: Ispis podataka za poligon 3

Iz polarnog dijagrama vidljivo je da su prikazana 4 prolaza i da je crvena kutija bliža donjem desnom kutu. Crvena kutija spaja se u jednu prepreku sa zidom zbog toga što je udaljenost tih dviju točaka manja od 300 mm.

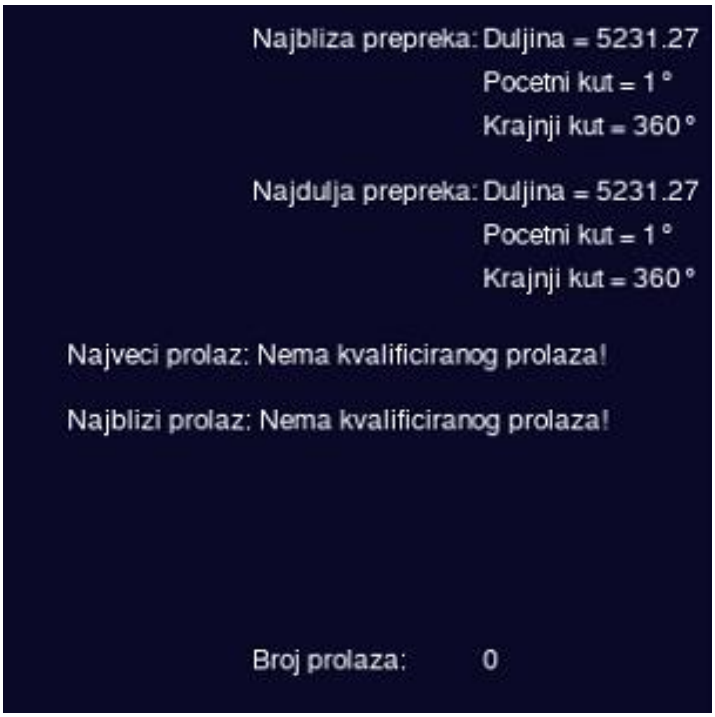
7.4. Poligon 4



Slika 25: Poligon 4



Slika 26: Polarni dijagram poligona 4



Slika 27: Ispis podataka za poligon 4

U poligonu 4 je LiDAR skener okružen preprekama kako bi se ispitao i taj granični slučaj. Kao što se vidi u Slika 26, nema ni jednog prolaza i sve se prepreke definiraju kao jedna, najveća. U ispisu podataka (Slika 27) prikazano je da je broj prolaza jednak nuli i da nema kvalificiranog prolaza koji bi se ispisivao na mjestu najvećeg i najbližeg prolaza. Prepreka je definirana kao jedan komad bez prekida.

8. ZAKLJUČAK

U današnje vrijeme velika je potreba za uređajima koji određuju položaj u prostoru, bio to GPS, radar, sonar ili pak LiDAR. Ti se uređaji koriste u svim mogućim uređajima poput mobilnih telefona, robotskih usisavača i kosilica, automobila, električnih invalidskih kolica i mnogih drugih, no njihova je primjena vrlo bitna u mobilnoj robotici. Mobilni se roboti kreću kroz nepredvidiv prostor, često sa zadaćom koju je potrebno ispuniti, te im je pomoć u navigaciji i planiranju trajektorije dobro došla za minimiziranje neželjenih troškova.

U prvom dijelu rada glavni zadatak bio je prikazati i opisati tehnologiju za razvoj sustava koji će modelirati okolinu mobilnog robota, a to su RPLidar A2M8, Raspberry Pi 2 mikro-računalo, osobno računalo i sitni pribor (različiti kablovi). Svaka navedena tehnologija ima svoje kvalitetnije i skuplje nasljednike i ovaj završni rad rađen je s ciljem da bude što pristupačniji i kompatibilan s više različitih kombinacija uređaja s minimalnom izmjenom programskog koda.

U drugom dijelu rada bilo je potrebno prikazati rješenje u obliku algoritama implementiranih u programski kod, tj. izlučiti podatke koje daje LiDAR skener, obraditi ih i prikazati u polarnom dijagramu i ispisati informacije o njima. S obzirom da je trebalo filtrirati ulazne podatke koje daje skener i iz njih izlučiti prepreke, prolaze i slično, zaključeno je da je to vrlo zahtjevan proces za mikro-računalo i korišteni su što jednostavniji i brži algoritmi. Ti su se algoritmi mogli na lak način poboljšati, no za to je potrebno bolje hardversko okruženje, odnosno Raspberry Pi 4 mikro-računalo koje je otprilike 3 puta brže od korištenog. Također, zapaženo je da je učestalost anomalija veća što je prepreka udaljenija zbog toga što jedna zraka pokriva veći dio prostora i može lakše doći do disperzije svjetlosne zrake jer prostor koji pokriva jedan stupanj na 100 mm nije jednak onome koji pokriva na 1000 mm. Moguća rješenja za taj problem su implementacija boljeg filtracijskog sistema anomalija, bolji LiDAR skener - ili oboje, no gledano iz financijskog aspekta najbolje je rješenje optimizacija algoritma za filtriranje anomalija.

LITERATURA

[1] Moubarak P., Ben-Tzvi P.: *Adaptive Manipulation of a Hybrid Mechanism Mobile Robot*, IEEE International Symposium on Robotic and Sensors Environments (ROSE), 2011.

[2] Fuentes-Pacheco J., Ruiz-Ascencio J., Rendón-Mancha J. M.: *Visual simultaneous localization and mapping: a survey*, 2005.

[3] Gajski, D.: *Osnove laserskog skeniranja iz zraka*, Znanstveni časopis Geodetskog fakulteta Ekscentar, 2007.

[4] Lefsky, M.A., Cohen, W.B., Parker, G.G., Harding, D.J.: *Lidar Remote Sensing for Ecosystem Studies*. BioScience 52, 2002.

[5] Shanghai Slamtec.Co, Ltd: *RPLiDAR A2 Introduction and datasheet*, 2017.

[6] Raspberry Pi Foundation: *What is a Raspberry Pi?*, Pristupljeno 08.siječnja 2021., https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi

[7] Raspberry Pi Foundation:*Raspberry Pi OS (64-bit) beta test version*, Pristupljeno 2. veljače 2021., https://www.raspberrypi.org/software/

[8] Raspberry Pi Foundation: *GPIO*, Pristupljeno 5. veljače 2021., https://www.raspberrypi.org/documentation/usage/gpio

[9] Kuhlman, D.: *A Python Book: Beginning Python, Advanced Python and Python Exercises*, Poglavlje 1.1, 2012.

[10] Python Software Foundation: *Python Developers Guide*, Pristupljeno 01. veljače 2021., https://devguide.python.org/

[11] Python Software Foundation: *PEP 20 -- The Zen of Python*, Pristupljeno 1. veljače 2021., https://www.python.org/dev/peps/pep-0020/

[12] Python Software Foundation: *Is Python a good language for beginning programmers?*, Pristupljeno 1. veljače 2021., https://docs.python.org/3/faq

[13] Microsoft Visual Studio Code: *FAQ*, Pristupljeno 7. veljače 2021., https://code.visualstudio.com/docs/supporting/faq

[14] Stack Overflow: *Developer Surver Results – Most Popular Development Environment*, 2019., https://insights.stackoverflow.com/survey/2019#development-environments-and-tools

PRILOG

I. Python programski kod

i. *lidar.py*

ii. *anomalija_filter.py*

iii. *sensor_adapter.py*

iv. *tocka.py*

v. *udaljenost_tocke.py*

vi. *prepreka.py*

vii. *prolaz.py*

viii. *pomagala.py*

ix. *ekran.py*

lidar.py

```
import os
from math import cos, sin, pi, floor
import pygame
from adafruit_rplidar import RPLidar
import time
from ekran import Ekran
from sensor_adapter import lista_podataka_u_listu_tocaka
from pomagala import grupiraj_prepreke
from tocka import Tocka
import anomalija_filter

broj_avg_podataka = 4
scan_data = [None]*broj_avg_podataka
ekran = Ekran()
max_udaljenost_tocke = 300

scan_index = 0

lidar = RPLidar(None, '/dev/ttyUSB0')
time.sleep(2)

def lidar_scans(self, max_buf_meas=360, min_len=360):

        lidar.stop()
        scan = []
        iterator = lidar.iter_measurments(max_buf_meas)
        for new_scan, quality, angle, distance in iterator:
            if new_scan:
                if len(scan) > min_len:
                    yield scan
                    scan = []
            if quality > 0 and distance > 0:
                scan.append((quality, angle, distance))

def sort_by_angle(tocka: Tocka):
    return tocka.angle


for scan in lidar_scans(lidar):
    if ekran.provjeri_exit():
        # zaustavi lidar
        lidar.set_pwm(0)
        # odspoji lidar
        lidar.stop()
        lidar.disconnect()
        # ugasi program
        ekran.exit()
```

anomalija_filter.py

```
from tocka import Tocka
from pomagala import udaljenost_dvije_tocke

treshold_tocke = 4

def dohvati_srednju_vrijednost_tocaka(polje_skenova: []):

    map_tocke_na_istom_kutu = dict()

    for i in range(0, len(polje_skenova)):
        scan_podaci = polje_skenova[i]

        for tocka in scan_podaci:
            kut = round(tocka.angle)
            if (kut in map_tocke_na_istom_kutu) == False:
                map_tocke_na_istom_kutu[kut] = []
            map_tocke_na_istom_kutu[kut].append(tocka.distance)

    avg_vrijednosti_po_kutu = []
    for kut in map_tocke_na_istom_kutu:
        tocke = map_tocke_na_istom_kutu[kut]
        if len(tocke) < treshold_tocke:
            continue

        suma = sum(tocke) - min(tocke) - max(tocke)
        broj_tocaka = len(tocke) - 2
        avg_dist = suma / broj_tocaka
        avg_tocka = Tocka(kut, avg_dist)
        avg_vrijednosti_po_kutu.append(avg_tocka)

    return avg_vrijednosti_po_kutu
```

sensor_adapter.py

```
from tocka import Tocka

def podatak_senzora_u_tocku(sensor_data: (float, float, float)) -> Tocka:
    (_, angle, distance) = sensor_data
    return Tocka(angle, distance)

def lista_podataka_u_listu_tocaka(data: []) -> []:
    lista_tocaka = []
    for sensor_data in data:
        tocka = podatak_senzora_u_tocku(sensor_data)
        lista_tocaka.append(tocka)

    return lista_tocaka
```

tocka.py

```
class Tocka:
    angle = 0.0
    distance = 0.0

    def __init__(self, angle: float, distance: float):
        '''
        Inicijalizira tocku
        '''
        self.angle = angle
        self.distance = distance

    def __str__(self):
        '''
        Vraca objekt u string formatu (npr. str(tocka))
        '''
        return "[A: " + str(self.angle) + "; D: " + str(self.distance) + "]"

    def __repr__(self):
        '''
        Poziva se prilikom potrebe za prikaz objekta (ispis liste, debug..)
        '''
        return str(self)
```

udaljenost_tocke.py

```
from tocka import Tocka
from math import radians, sin, cos, sqrt, pow

def udaljenost_dvije_tocke(tocka1: Tocka, tocka2: Tocka) -> float:
    '''
    Racuna udaljenost dvije tocke u polarnom koordinatnom sustavu
    '''
    if tocka1.distance < tocka2.distance:
        (tocka1, tocka2) = (tocka2, tocka1)

    r1 = tocka1.distance
    r2 = tocka2.distance
    kut = radians(abs(tocka1.angle - tocka2.angle))

    r2y = r2 * cos(kut)
    r2x = r2 * sin(kut)
    q = r1 - r2y
    udaljenost = sqrt(pow(r2x, 2) + pow(q, 2))

    return udaljenost
```

prepreka.py

```
from tocka import Tocka
from udaljenost_tocke import udaljenost_dvije_tocke

def uzmi_najblizu_tocku(tocka1: Tocka, tocka2: Tocka) -> Tocka:
    if tocka1 == None:
        return tocka2
    elif tocka2 == None:
        return tocka1

    if tocka1.distance < tocka2.distance:
        return tocka1
    else:
        return tocka2

class Prepreka:
    tocke = []
    duljina: float = 0
    najbliza_tocka: Tocka = None

    def __init__(self):
        '''
        Inicijalizira prepreku
        '''
        self.tocke = []

    def dodaj_tocku(self, tocka: Tocka):
        '''
        Dodaje tocku u prepreku
        '''
        # Povecaj duljinu prepreke
        predhodna_tocka = self.dohvati_zadnju_tocku()
        if predhodna_tocka != None:
            self.duljina += udaljenost_dvije_tocke(predhodna_tocka, tocka)

        # Azuriraj najblizu tocku prepreke
        self.najbliza_tocka = uzmi_najblizu_tocku(self.najbliza_tocka, tocka)

        # Dodaj tocku u listu
        self.tocke.append(tocka)

    def dohvati_tocke(self) -> []:
        '''
        Dohvaca sve tocke iz prepreke
        '''
        return self.tocke
```

```
    def broj_tocaka(self) -> int:
        '''
        Vraca broj tocaka u prepreci
        '''
        return len(self.tocke)

    def dohvati_prvu_tocku(self) -> Tocka:
        '''
        Dohvaca prvu dodanu tocku u prepreku
        '''
        if (self.broj_tocaka() < 1):
            return None

        return self.tocke[0]

    def dohvati_zadnju_tocku(self) -> Tocka:
        '''
        Dohvaca zadnje dodanu tocku u prepreku
        '''
        if (self.broj_tocaka() < 1):
            return None

        return self.tocke[self.broj_tocaka() - 1]

    def uvazavajuca_prepreka(self) -> bool:
        '''
        Provjerava da li prepeka zadovoljava uvjete da bi se kvalificirala kao
prepreka, tj. ako ima vise od 2 tocke
        '''
        return self.broj_tocaka() > 2

    def __str__(self):
        '''
        Vraca objekt u string formatu (npr. str(prepreka))
        '''
        return  str(self.tocke)

    def __repr__(self):
        '''
        Poziva se prilikom potrebe za prikaz objekta (ispis liste, debug..)
        '''
        return str(self)
```

prolaz.py

```
from udaljenost_tocke import udaljenost_dvije_tocke
from prepreka import Prepreka
from tocka import Tocka


class Prolaz:
    pocetna_tocka: Tocka = None
    krajnja_tocka: Tocka = None
    velicina_prolaza: float = 0

    def __init__(self, prepreka1: Prepreka, prepreka2: Prepreka):
        self.pocetna_tocka = prepreka1.dohvati_zadnju_tocku()
        self.krajnja_tocka = prepreka2.dohvati_prvu_tocku()
        self.velicina_prolaza = udaljenost_dvije_tocke(self.pocetna_tocka, sel
f.krajnja_tocka)

    def dohvati_tocku_sredine_prolaza(self) -> Tocka:
        srednji_kut = (self.pocetna_tocka.angle + self.krajnja_tocka.angle) /
2
        srednja_udaljenost = (self.pocetna_tocka.distance + self.krajnja_tocka
.distance) / 2
        return Tocka(srednji_kut, srednja_udaljenost)

    def __str__(self):
        '''
        Vraca objekt u string formatu (npr. str(tocka))
        '''
        return "[S: " + str(self.velicina_prolaza) +"]"

    def __repr__(self):
        '''
        Poziva se prilikom potrebe za prikaz objekta (ispis liste, debug..)
        '''
        return str(self)

def ekstrakcija_prolaza(lista_prepreka: [], min_velicina_prolaza: int, max_vel
icina_prolaza: int) -> []:
    prolazi = []
    for i in range(len(lista_prepreka) - 1):
        prepreka1 = lista_prepreka[i]
        prepreka2 = lista_prepreka[i + 1]
        prolaz = Prolaz(prepreka1, prepreka2)
        if prolaz.velicina_prolaza >= min_velicina_prolaza and prolaz.velicina
_prolaza <= max_velicina_prolaza:
            prolazi.append(prolaz)

    prolaz = Prolaz(lista_prepreka[len(lista_prepreka) - 1], lista_prepreka[0])
```

```
    if prolaz.velicina_prolaza >= min_velicina_prolaza and prolaz.velicina_pro
laza <= max_velicina_prolaza:
        prolazi.append(prolaz)

    return prolazi

def dohvati_najveci_prolaz(lista_prolaza: []) -> Prolaz:
    if len(lista_prolaza) == 0:
        return None

    najveci_prolaz = lista_prolaza[0]

    for prolaz in lista_prolaza:
        if najveci_prolaz.velicina_prolaza < prolaz.velicina_prolaza:
            najveci_prolaz = prolaz

    return najveci_prolaz


def dohvati_najblizi_prolaz(lista_prolaza: []) -> Prolaz:
    if len(lista_prolaza) == 0:
        return None

    najblizi_prolaz = lista_prolaza[0]

    for prolaz in lista_prolaza:
        if najblizi_prolaz.dohvati_tocku_sredine_prolaza().distance > prolaz.d
ohvati_tocku_sredine_prolaza().distance:
            najblizi_prolaz = prolaz

    return najblizi_prolaz
```

pomagala.py

```
from tocka import Tocka
from prepreka import Prepreka
from udaljenost_tocke import udaljenost_dvije_tocke

def grupiraj_prepreke(tocke: [], max_udaljenost_tocke: float) -> []:
    '''
    tocke: Lista tocaka
    return: Lista prepreka
    '''
    lista_prepreka = []
    prepreka = Prepreka()

    for i in range(len(tocke) - 1):
        tocka1: Tocka = tocke[i]
        tocka2: Tocka = tocke[i + 1]

        if prepreka.broj_tocaka() == 0:
            prepreka.dodaj_tocku(tocka1)
        udaljenost_tocke = udaljenost_dvije_tocke(tocka1, tocka2)
        if udaljenost_tocke < max_udaljenost_tocke:
            prepreka.dodaj_tocku(tocka2)
        else:
            if prepreka.uvazavajuca_prepreka():
                lista_prepreka.append(prepreka
            prepreka = Prepreka()
    if prepreka.uvazavajuca_prepreka():
        lista_prepreka.append(prepreka)

    return lista_prepreka

def dohvati_najdulju_prepreku(lista_prepreka: []) -> Prepreka:
    najdulja_prepreka = lista_prepreka[0]

    for prepreka in lista_prepreka:
        if najdulja_prepreka.duljina < prepreka.duljina:
            najdulja_prepreka = prepreka

    return najdulja_prepreka


def dohvati_najblizu_prepreku(lista_prepreka: []) -> Prepreka:
    najbliza_prepreka = lista_prepreka[0]

    for prepreka in lista_prepreka:
        if najbliza_prepreka.najbliza_tocka.distance > prepreka.najbliza_tocka
.distance:
            najbliza_prepreka = prepreka
    return najbliza_prepreka
```

ekran.py

```
import pygame
import time
import random
from tocka import Tocka
from prepreka import Prepreka
from pomagala import grupiraj_prepreke, dohvati_najdulju_prepreku, dohvati_naj
blizu_prepreku
from math import sin, cos, radians, pi
import prolaz

DARK_BLUISH = (10, 10, 40)
RED = (255, 10, 40)
BLUE = (10, 10, 255)
GREEN = (10, 255, 10)
WHITE_COLOR = pygame.Color(255, 255, 255)
WHITE = (255, 255, 255)

VIOLETISH = (100, 149, 237)
YELLOWISH = (255, 191, 0)
PINKISH = (222, 49, 99)


class Ekran:
    width: int = 1280
    height: int = 720
    max_distance: int = 3000
    background_boja = DARK_BLUISH

    najbliza_prepreka_boja = YELLOWISH
    najdulja_prepreka_boja = VIOLETISH
    najdulja_i_najbliza_prepreka_boja = VIOLETISH
    default_boja_prepreke = WHITE

    moguci_prolaz_boja = GREEN

    size = (width, height)
    font = None

    def __init__(self):
        '''
        Inicijalizira ekran
        '''
        pygame.init()
        pygame.display.init()
        pygame.display.set_caption('Polar diagram')

        self.screen = pygame.display.set_mode(self.size)
        self.font = pygame.font.SysFont('Arial', 12)
```

```
def tocka_u_koordinate(self, tocka: Tocka) -> (int, int):
        '''
        Prebacuju polarne koordinate u kartezijev koordinatni sustav
        '''
        kut = radians(tocka.angle)
        x = tocka.distance * cos(kut)
        y = tocka.distance * sin(kut)
        point = (int(self.width / 2) + int(x / self.max_distance * (self.width
 / 2)), int(self.height / 2) + int(y / self.max_distance * (self.height / 2)))

        return point

    def nacrtaj_koordinate(self):
        '''
        Crta X i Y koordinatne osi na ekranu
        '''
        pygame.draw.line(self.screen, WHITE_COLOR, (0, int(self.height/2)), (s
elf.width, int(self.height/2)))
        pygame.draw.line(self.screen, WHITE_COLOR, (int(self.width/2), 0), (in
t(self.width/2), self.height))

    def crtanje_i_spajanje_dviju_tocaka(self, tocka1: Tocka, tocka2: Tocka, bo
ja: pygame.Color):
        point_start: (int, int) = self.tocka_u_koordinate(tocka1)
        point_end: (int, int) = self.tocka_u_koordinate(tocka2)

        pygame.draw.circle(self.screen, boja, point_start, 2)
        pygame.draw.circle(self.screen, boja, point_end, 2)
        pygame.draw.line(self.screen, boja, point_start, point_end)

    def nacrtaj_prepreku(self, prepreka: Prepreka, boja: pygame.Color):
        for i in range(len(prepreka.tocke) - 1):
            tocka_start = prepreka.tocke[i]
            tocka_end = prepreka.tocke[i + 1]
            self.crtanje_i_spajanje_dviju_tocaka(tocka_start, tocka_end, boja)


    def nacrtaj_prepreke(self, lista_prepreka: []):
        '''
        Crta dobivene prepreke na ekranu
        '''
        najbliza_prepreka = dohvati_najblizu_prepreku(lista_prepreka)
        najdulja_prepreka = dohvati_najdulju_prepreku(lista_prepreka)

        for prepreka in lista_prepreka:
            boja_prepreke = self.default_boja_prepreke
            if prepreka == najbliza_prepreka and prepreka == najdulja_prepreka
:
                boja_prepreke = self.najdulja_i_najbliza_prepreka_boja
```

```
            elif prepreka == najbliza_prepreka:
                boja_prepreke = self.najbliza_prepreka_boja
            elif prepreka == najdulja_prepreka:
                boja_prepreke = self.najdulja_prepreka_boja

            self.nacrtaj_prepreku(prepreka, boja_prepreke )

    def nacrtaj_prolaze(self, prolazi: []):
        '''
        Crta dobivene prolaze na ekranu
        '''
        for prolaz in prolazi:
            tocka_start = prolaz.pocetna_tocka
            tocka_end = prolaz.krajnja_tocka

            self.crtanje_i_spajanje_dviju_tocaka(tocka_start, tocka_end, self.
moguci_prolaz_boja)

    def legenda(self):
        '''
        Crta legendu boja
        '''
        # Najbliza prepreka
        pygame.draw.circle(self.screen, YELLOWISH, (10, 15), 2)
        pygame.draw.circle(self.screen, YELLOWISH, (50, 15), 2)
        pygame.draw.line(self.screen, YELLOWISH, (10, 15), (50, 15))
        najbliza_prepreka = self.font.render('Najbliza prepreka', True, WHITE)
        self.screen.blit(najbliza_prepreka, (70, 7))

        # Najdulja prepreka
        pygame.draw.circle(self.screen, VIOLETISH, (10, 35), 2)
        pygame.draw.circle(self.screen, VIOLETISH, (50, 35), 2)
        pygame.draw.line(self.screen, VIOLETISH, (10, 35), (50, 35))
        najdulja_prepreka = self.font.render('Najdulja prepreka', True, WHITE)
        self.screen.blit(najdulja_prepreka, (70, 27))

        # Zeleni prolaz
        pygame.draw.circle(self.screen, GREEN, (10, 55), 2)
        pygame.draw.circle(self.screen, GREEN, (50, 55), 2)
        pygame.draw.line(self.screen, GREEN, (10, 55), (50, 55))
        zeleni_prolaz = self.font.render('Prolaz 500-1000 mm', True, WHITE)
        self.screen.blit(zeleni_prolaz, (70, 47))
```

```
    def informacije(self, lista_prepreka: [], prolazi: []):
        '''
        Ispisuje izlucene informacije
        '''
        # Najbliza prepreka
        najbliza_prepreka = dohvati_najblizu_prepreku(lista_prepreka)
        duljina_najblize_prepreke = str(round(najbliza_prepreka.duljina, 2))
        text_najbliza_prepreka = 'Najbliza prepreka: '
        render_text = self.font.render(text_najbliza_prepreka, True, WHITE)
        self.screen.blit(render_text, (1100, 15))
        text_najbliza_prepreka = 'Duljina = ' + duljina_najblize_prepreke
        render_text = self.font.render(text_najbliza_prepreka, True, WHITE)
        self.screen.blit(render_text, (1190, 15))

        prva_tocka = najbliza_prepreka.dohvati_prvu_tocku()
        zadnja_tocka = najbliza_prepreka.dohvati_zadnju_tocku()

        kut_prve_tocke = str(round(prva_tocka.angle, 2))
        kut_zadnje_tocke = str(round(zadnja_tocka.angle, 2))

        text_najbliza_prepreka_prvi_kut = 'Pocetni kut = ' + kut_prve_tocke +
'°'
        text_najbliza_prepreka_krajnji_kut = 'Krajnji kut = ' + kut_zadnje_toc
ke + '°'

        render_text = self.font.render(text_najbliza_prepreka_prvi_kut, True,
WHITE)
        self.screen.blit(render_text, (1190, 35))
        render_text = self.font.render(text_najbliza_prepreka_krajnji_kut, Tru
e, WHITE)
        self.screen.blit(render_text, (1190, 55))

        # Najveca prepreka
        najdulja_prepreka = dohvati_najdulju_prepreku(lista_prepreka)
        duljina_najdulje_prepreke = str(round(najdulja_prepreka.duljina, 2))
        text_najdulja_prepreka = 'Najdulja prepreka: '
        render_text = self.font.render(text_najdulja_prepreka, True, WHITE)
        self.screen.blit(render_text, (1100, 85))
        text_najdulja_prepreka = 'Duljina = ' + duljina_najdulje_prepreke
        render_text = self.font.render(text_najdulja_prepreka, True, WHITE)
        self.screen.blit(render_text, (1190, 85))

        prva_tocka = najdulja_prepreka.dohvati_prvu_tocku()
        zadnja_tocka = najdulja_prepreka.dohvati_zadnju_tocku()

        kut_prve_tocke = str(round(prva_tocka.angle, 2))
        kut_zadnje_tocke = str(round(zadnja_tocka.angle, 2))
```

```
        text_najdulja_prepreka_prvi_kut = 'Pocetni kut = ' + kut_prve_tocke +
'°'
        text_najdulja_prepreka_krajnji_kut = 'Krajnji kut = ' + kut_zadnje_toc
ke + '°'

        render_text = self.font.render(text_najdulja_prepreka_prvi_kut, True,
WHITE)
        self.screen.blit(render_text, (1190, 105))
        render_text = self.font.render(text_najdulja_prepreka_krajnji_kut, Tru
e, WHITE)
        self.screen.blit(render_text, (1190, 125))

        # Najveci prolaz
        najveci_prolaz = prolaz.dohvati_najveci_prolaz(prolazi)
        if najveci_prolaz != None:
            duljina_najveceg_prolaza = str(round(najveci_prolaz.velicina_prola
za, 2))
            text_najveci_prolaz = 'Najveci prolaz: '
            render_text = self.font.render(text_najveci_prolaz, True, WHITE)
            self.screen.blit(render_text, (1100, 160))
            text_najveci_prolaz = 'Duljina = ' + duljina_najveceg_prolaza
            render_text = self.font.render(text_najveci_prolaz, True, WHITE)
            self.screen.blit(render_text, (1190, 160))

            prva_tocka = najveci_prolaz.pocetna_tocka
            zadnja_tocka = najveci_prolaz.krajnja_tocka

            kut_prve_tocke = str(round(prva_tocka.angle, 2))
            kut_zadnje_tocke = str(round(zadnja_tocka.angle, 2))

            text_najveci_prolaz_prvi_kut = 'Pocetni kut = ' + kut_prve_tocke +
 '°'
            text_najveci_prolaz_krajnji_kut = 'Krajnji kut = ' + kut_zadnje_to
cke + '°'

            render_text = self.font.render(text_najveci_prolaz_prvi_kut, True,
 WHITE)
            self.screen.blit(render_text, (1190, 180))
            render_text = self.font.render(text_najveci_prolaz_krajnji_kut, Tr
ue, WHITE)
            self.screen.blit(render_text, (1190, 200))
        else:
            text_najveci_prolaz = 'Najveci prolaz: Nema kvalificiranog prolaza
!'
            render_text = self.font.render(text_najveci_prolaz, True, WHITE)
            self.screen.blit(render_text, (1100, 160))
```

```
        # Najblizi prolaz
        najblizi_prolaz = prolaz.dohvati_najblizi_prolaz(prolazi)
        if najblizi_prolaz != None:
            duljina_najblizeg_prolaza = str(round(najblizi_prolaz.velicina_pro
laza, 2))
            text_najblizi_prolaz = 'Najblizi_prolaz: '
            render_text = self.font.render(text_najblizi_prolaz, True, WHITE)
            self.screen.blit(render_text, (1100, 230))
            text_najblizi_prolaz = 'Duljina = ' + duljina_najblizeg_prolaza
            render_text = self.font.render(text_najblizi_prolaz, True, WHITE)
            self.screen.blit(render_text, (1190, 230))

            prva_tocka = najblizi_prolaz.pocetna_tocka
            zadnja_tocka = najblizi_prolaz.krajnja_tocka

            kut_prve_tocke = str(round(prva_tocka.angle, 2))
            kut_zadnje_tocke = str(round(zadnja_tocka.angle, 2))

            text_najblizi_prolaz_prvi_kut = 'Pocetni kut = ' + kut_prve_tocke
+ '°'
            text_najblizi_prolaz_krajnji_kut = 'Krajnji kut = ' + kut_zadnje_t
ocke + '°'

            render_text = self.font.render(text_najblizi_prolaz_prvi_kut, True
, WHITE)
            self.screen.blit(render_text, (1190, 250))
            render_text = self.font.render(text_najblizi_prolaz_krajnji_kut, T
rue, WHITE)
            self.screen.blit(render_text, (1190, 270))
        else:
            text_najblizi_prolaz = 'Najblizi prolaz: Nema kvalificiranog prola
za!'
            render_text = self.font.render(text_najblizi_prolaz, True, WHITE)
            self.screen.blit(render_text, (1100, 190))

        # Broj prolaza
        broj_prolaza = str(len(prolazi))
        render_text = self.font.render('Broj prolaza: ', True, WHITE)
        self.screen.blit(render_text, (1100, 300))
        render_text = self.font.render(broj_prolaza, True, WHITE)
        self.screen.blit(render_text, (1190, 300))


    def update(self, lista_prepreka: []):
        '''
        Azurira stanje na ekranu
        '''
        min_velicina_prolaza = 500
        max_velicina_prolaza = 1500
```

```
        prolazi = prolaz.ekstrakcija_prolaza(lista_prepreka, min_velicina_prol
aza, max_velicina_prolaza)

        self.screen.fill(self.background_boja)

        self.nacrtaj_koordinate()
        self.nacrtaj_prepreke(lista_prepreka)
        self.nacrtaj_prolaze(prolazi)
        self.legenda()
        self.informacije(lista_prepreka, prolazi)

        pygame.display.update()

    def provjeri_exit(self) -> bool:
        '''
        Vraca True ako korisnik zeli zatvoriti prozor
        '''
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                return True
        return False

    def exit(self):
        '''
        Zatvara prozor i izlazi iz programa
        '''
        pygame.quit()
        exit()
```