

Univerzitet “Džemal Bijedić”

Fakultet Informacijskih Tehnologija

ZAVRŠNI RAD

Izbjegavanje prepreka u mobilnoj robotici korištenjem LIDAR-a i A\* algoritma

Profesor: Student: vanr. prof. Elmir Babović Adil Eminagić, IB200200

Akademska godina: 2024/2025

Mostar, 2024. godina

IZJAVA O AUTORSTVU

Ja, ADIL (SAŠA) EMINAGIĆ, student Fakulteta informacijskih tehnologija, Univerziteta "Džemal Bijedić" u Mostaru, pod punom moralnom, materijalnom i krivičnom odgovornošću,

Izjavljujem

da je rad pod naslovom

Izbjegavanje prepreka u mobilnoj robotici korištenjem LIDAR-a i A\* algoritma

u potpunosti rezultat sopstvenog istraživanja, gdje su korišteni sadržaji (tekst, ilustracije, tabele itd.) drugih autora jasno označeni pozivanjem na izvor i ne narušavaju bilo čija vlasnička ili autorska prava.

U Mostaru, 02.11.2024. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Adil Eminagić, IB200200

[Uvod 1](#_Toc180707170)

[Apstrakt 2](#_Toc180707171)

[Tehnologije i uređaji korišteni u radu 3](#_Toc180707172)

[Tehnologije korištene u radu 3](#_Toc180707173)

[C# programski jezik i Windows Forms aplikacija 3](#_Toc180707174)

[Python programski jezik 3](#_Toc180707175)

[Lidar (Light Detection and Ranging) 4](#_Toc180707176)

[Uređaji korišteni u radu 5](#_Toc180707177)

[Arduino 5](#_Toc180707178)

[Koračni motori 6](#_Toc180707179)

[Mikrostep drajveri 7](#_Toc180707180)

[Napojna jedinica 8](#_Toc180707181)

[Pronalazak najkraće putanje 9](#_Toc180707182)

[Uspostava serijeske komunikacije sa Arduinom 9](#_Toc180707183)

[Plan prvog rješenja 11](#_Toc180707184)

[Komunikacija Lidara i C# Windows Forms aplikacije 12](#_Toc180707185)

[Serijska komunikacija sa Lidarom 12](#_Toc180707186)

[Pretvaranje podataka u data pakete 14](#_Toc180707187)

[Nedostaci prvog rješenja i novo rješenje 19](#_Toc180707188)

[Komunikacija Lidara, Python programa i C# Windows Forms aplikacije 20](#_Toc180707189)

[Prikaz mapiranog prostora u Python programu 23](#_Toc180707190)

[Priprema podataka za algoritam pretrage 24](#_Toc180707191)

[Algoritam pretrage 27](#_Toc180707192)

[A\* algoritam pretrage 27](#_Toc180707193)

[Testiranje algoritma 27](#_Toc180707194)

[Primjena algoritma 31](#_Toc180707195)

[Kreiranje izvšnog dijela mobilnog robota 34](#_Toc180707196)

[Sklapanje i povezivanje fizičkih komponenti 34](#_Toc180707197)

[Spajanje koračnih motora sa mikrostep drajverima 34](#_Toc180707198)

[Spajanje mikrostep drajvera sa Arduinom 35](#_Toc180707199)

[Kod za kretanje koračnih motora u Arduino razvojnom okruženju 36](#_Toc180707200)

[Uspostavljanje komunikacije između koračnih motora 36](#_Toc180707201)

[Definisanje komadi za kretanje mobilnog robota 37](#_Toc180707202)

[Određivanje dužine kretanja robota 40](#_Toc180707203)

[Računanje vremena na osnovu distance za pravolinijska kretanja 40](#_Toc180707204)

[Računanje vremena za rotaciju mobilog robota 41](#_Toc180707205)

[Upravljanje višestrukim komandama za kretanje 41](#_Toc180707206)

[Integracija svih dijelova mobilnog robota 42](#_Toc180707207)

[Prilagođavanje prostora 42](#_Toc180707208)

[Prilagođavanje putanje 45](#_Toc180707209)

[Prilagođavanje kompleksne putanje 49](#_Toc180707210)

[Zaključak 53](#_Toc180707211)

[Buduća unapređenja 53](#_Toc180707212)

[Literatura 54](#_Toc180707213)

## Uvod

U današnjem savremenom dobu, tehnološki porast je veći nego ikada, a mobilni roboti i umjetna i inteligencija postaju nezamjenjivi dijelovi u različitim industrijama i svakodnevnom životu. Mobilni roboti sa modernim sistemima umjetne inteligencije mogu samostalno obavljati zadatke, vršiti interakciju sa okolinom i na osnovu podataka sakupljenih iz okoline donositi zaključke, prilagođavati se brzo raznim uslovima. Sa svim tim karakteristikama roboti se koriste u raznim poljima automatske proizvodnje do medicinske robotike za usluge i prevoz.

Razvoj robotike započeo je 1950-ih godina, kada je napravljen prvi robot koji oponaša čovjeka kojeg su napravili. A danas zahvaljujući naglom razvoju tehnologije, roboti su postali sastavni dio industrije, i sve više postaju dio svakodnevnice. S obzirom na brzu digitalizaciju i porast potrebe za učinkovitim rješenjima, mobilna robotika i umjetna inteligencija stalno se razvijaju i unaprjeđuju. Korištenjem robota unaprjeđuje se produktivnost i otvaraju nova vrata inovacijama i unapređenjima životnog standarda. Osnovni cilj mobilnog robota je stići od početne do ciljne pozicije najkraćom putanjom izbjegavajući prepreke.

U ovom diplomskom radu bit će opisan čitav put do pravljenja i uspostavljanja mobilnog robota. Prije pravljenja robota nije sa znalo koje će se sve tehnologije koristiti niti je postojao jasan plan po kojem se odmah moglo preći na izradu, iz razloga što neke tehnologije koje su korištene nisu bile dovoljno istražene i nije bilo puno materija u kojima su objašnjene. Jedna od takvih tehnologija je Lidar (Light Detection and Ranging) tehnologija.

Put do konačnog rješenja također nije uvijek išao u pravom smjeru. Ostvarena su neka rješenja u određenim tehnologijama koja se na kraju nisu koristila. Ta rješenja će također biti prikazana, jer cilj je prikazati ne samo konačno rješenje, nego cijeli proces dolaska do tog rješenja uz sve prepreke, ograničenja i pogrešna rješenja.

Cilj ovog projekta je pokazati proces izrade mobilnog robota koji će se kretati po prostoru od početne do ciljne pozicije, izbjegavajući prepreke i prilagođavajući se dimenzijama robota, i sve to raditi samostalno bez čovjekovog upravljanja u različitim okuženjima. U radu će biti detaljno opisan kod kao i fizički dijelovi robota. Pored uspjeha i rješenja predstavit će se i nedostaci kao i moguća unapređenja.

## Apstrakt

U ovom diplomskom radu opisan je mobilni robot koji uz pomoć Lidar tehnologije mapira prostor, zatim kroz programski jezik Python korištenjem umjetne inteligencije pronalazi najkraći put od početnog položaja do odredišta, izbjegavajući prepreke i prilagođavajući se dimenzijama robota. Kretanje se vrši preko koračnih motora koji dobijaju komande za kretanja sa Arduina. Ovaj rad se sastoji od dva dijela: programskog koda i fizičkih dijelova. Da bi robot pravilno izvršavao sve potrebne zadatke potrebna je integracija oba dijela.

Fizički dijelovi su Lidar, Arduino mikrokontroler, koračni motori, mikrostep drajveri koji kontrolišu koračne motore, laptop i napojna jedinica. Svi ti dijelovi moraju biti pravilo povezani žicama i priključcima. Iako je sastavljanje tih dijelova na malom prostor koji se može kretati bilo jako izazovno, napravljena je ploča na kojoj su pričvršćeni točkovi i na koju se mogu staviti svi ostali dijelovi.

Prvobitni plan je bio da se za pronalazak najkraće putanje i komunikaciju između Lidara i Arduina koriti C# Windows Forms desktop aplikacija. Ostvarena su određena rješenja kao što je čitanje data paketa dobijenih od Lidara. Bila je ostvarena serijska komunikacija sa Lidarom, niz stringova je pretvoren u bajtove i paketi su bili prikazani. Bilo je potrebno izvršiti određene kalkulacije nad tim paketima, a u međuvremenu je pronađen Python kod koji izvršava potrebne kalkulacije, tako da se odustalo od ovog rješenja. Iako se to rješenje nije koristilo u radu, ono će biti prikazano jer prikazuje dohvaćanje i pretvaranje podataka dobijenih serijskom komunikacijom.

## Tehnologije i uređaji korišteni u radu

Da bi se ovaj rad lakše razumio, prvo je potrebno upoznati se sa uređajima i tehnologijama koje se koriste i koje su se koristile prilikom izrade mobilnog robota. Tehnologije koje su korištene: C# programski jezik unutar Windows Forms aplikacije, Python programski jezik i Lidar tehnologija. Od uređaja korišteni su koračni motori, Arduino Mega, mikrostep drajveri, napojna jedinica i Lidar (pored toga što je Lidar tehnologija, Lidar je također i uređaj).

### Tehnologije korištene u radu

Odabir tehnologija u ovom radu, a i generalno je jako važan, a to se posebno odnosi na odabir programskog jezika. Programski jezici imaju različite karakteristike, prednosti i nedostatke. U ovom radu potrebna je primjena umjetne inteligencije, jednostavan i čitljiv kod i vizualizacija, tako da su programski jezici visokog nivoa kao što su **C#** i **Python,** koji posjeduju biblioteke sa tim funkcionalnostima, dosta bolji izbor od programskih jezika koji nisu toliko jednostavni i ne posjeduju visok nivo apstrakcije. Za razliku od programskih jezika, Lidar tehnologija je sastavni dio mobilnog robota u ovom radu, tako da na nju nije trebalo paziti prilikom odabira, već samo istražiti.

#### C# programski jezik i Windows Forms aplikacija

**C#** je objektno orijentisani programski jezik visokog nivoa koji spada u porodicu C jezika zajedno sa programskim jezicima C i C++. Za razliku od C i C++, dosta je napredniji i jednostavniji za korištenje. Također posjeduje NuGet pakete koji su orijentisani na umjetnu inteligenciju i mašinsko učenje, kao što su Microsoft.ML i TensorFlow.NET, što je korisno za zadatke poput pronalaska najkraće putanje. U C#-u je prilično jednostavno ostvariti serijsku komunikaciju s vanjskim uređajima, a u ovom radu potrebno je uspostaviti serijsku komunikaciju s Lidarom i Arduinom.

**Windows Forms** je .NET biblioteka koja se koristi za razvijanje GUI (Graphical User Interface) desktop aplikacija. Kreiranje formi i dodavanje komponenti kao što su dugmad, tekstualna polja, padajući meniji je prilično jednostavno jer programeri mogu ručno pozicionirati komponente bez upotrebe programskog koda. Prostor koji Lidar mapira potrebno je vizuelno prikazati kao i putanju, tako da je Widnows Froms aplikacija dobar izbor za ovaj rad.

#### Python programski jezik

**Python** je programski jezik visokog nivoa, poznat po svojoj jednostavnosti i čitljivosti. Python je odličan izbor za početnike jer je lagan za učenje i posjeduje veliki broj ugrađenih funkcija, dok je istovremeno moćan alat za iskusne programere. Koristi se u raznim oblastima, uključujući web razvoj, analizu podataka, mašinsko učenje, umjetnu inteligenciju, automatizaciju zadataka, testiranje aplikacija, statistiku i vizualizaciju podataka. Python je često bolji izbor u poređenju s C# programskim jezikom, jer omogućava pisanje kraćeg i jednostavnijeg koda, a istovremeno je jedan od vodećih jezika u polju umjetne inteligencije i mašinskog učenja, što je ključno za izračunavanje putanja. Pored toga, Python posjeduje bogat ekosistem biblioteka koje omogućavaju jednostavnu vizualizaciju podataka, što je posebno važno jer jasna vizualizacija prostora ima značajnu ulogu u ovom radu.

#### Lidar (Light Detection and Ranging)

**Lidar (Light Deatection and Ranging)** je tehnologija i uređaj koji koristi laserske zrake kako bi odredio udaljenost tačaka od njega i time mapirao prostor. Lidar koji se koristi u ovom radu je LiDAR LD19, koji je proizvela kompanija LDROBOT. Ovaj Lidar mapira prostor za 360 stepeni oko sebe. Lidar emituje laserske zrake. Kada te zrake naiđu na prepreku one se odbiju od nju i vrate do Lidara, i na osnovu vremena koje je bilo potrebno da se zraka vrati, izračunava se udaljenost. Podaci koji se dobiju predstavljaju listu tačaka sa informacijama o uglu i distanci (npr. pod uglom od 45 stepeni, udaljenost do prepreke je 50 cm).

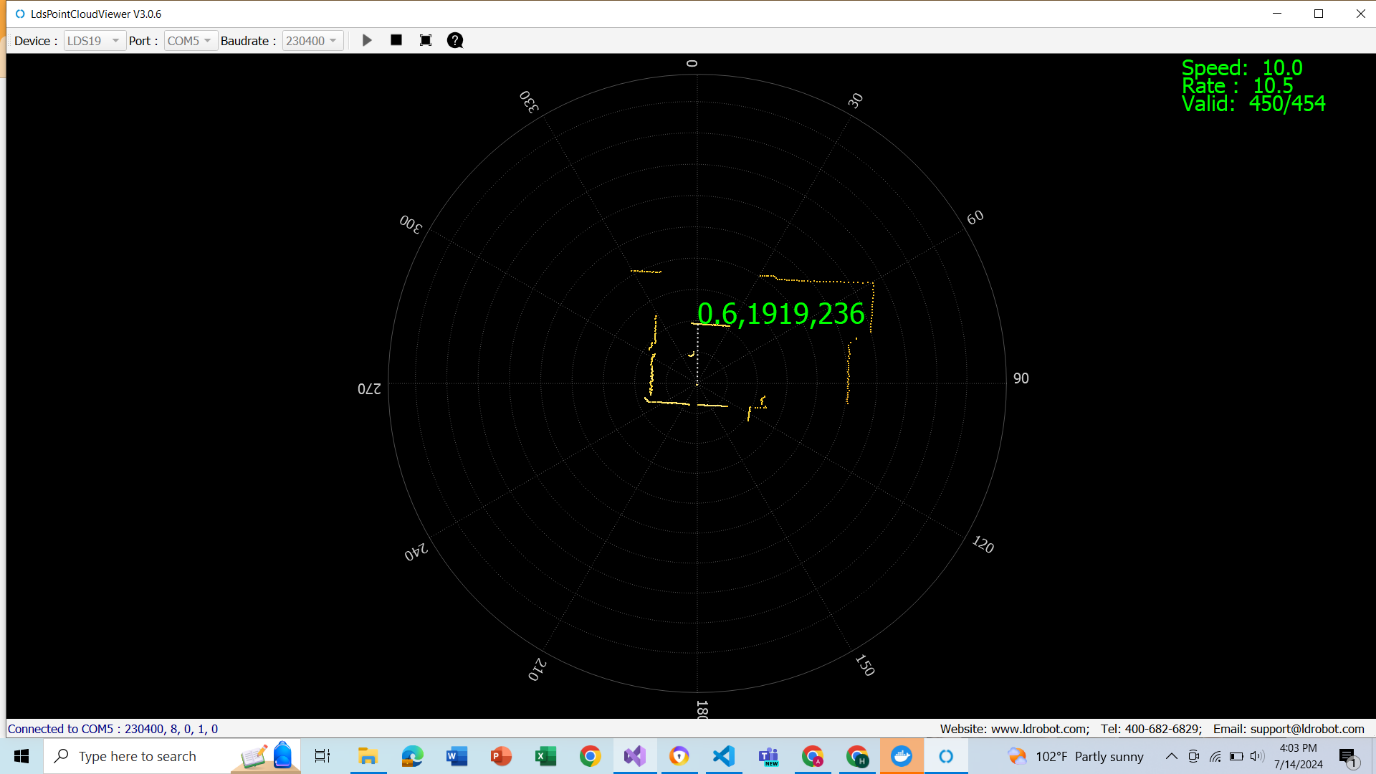
Lidar ima i svoje nedostatke. Iako mapira prostor za 360 stepeni oko sebe, Lidar prepoznaje prepreke samo na visini na kojoj se nalazi. Dakle ako je neka prepreka ispod ili iznad Lidara, ona neće biti prepoznata. Lidar prepoznaj samo prvu prepreku. Također ne može prepoznati dužinu prepreke, već samo da ta prepreka postoji.



Slika 1. LiDAR LD19

Postoji softver koji vizualizira mapirani prostor, i učitava promjene u realnom vremenu. Taj softver se naziva **LdsPointCloudViewer** softver koji se moze preuzeti na ovom linka <https://github.com/ldrobotSensorTeam/ld_desktop_tool/releases>.

Da bi se softver pokrenuo, potrebno je priključiti Lidar u računar ili laptop, i u konfiguraciji odabrati LDS19 uređaj i 230400 baud rate.



Slika 2. LdsPointCloudViewer softver

### Uređaji korišteni u radu

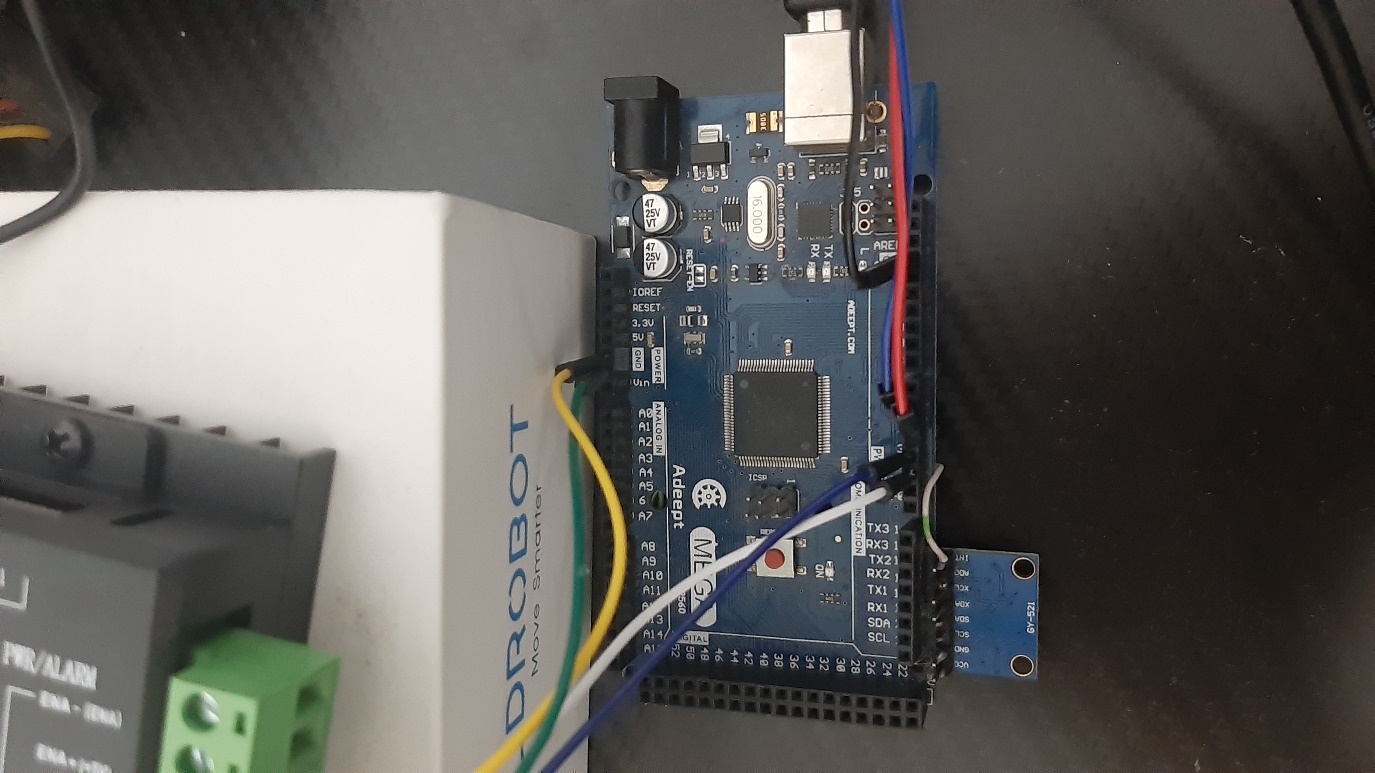
Pored tehnologija, jednako je važno i odabrati uređaje koji će se koristiti u radu. Uređaji trebaju biti kvalitetni i ispravni. Neispravnost uređaja može dovesti do grešaka u radu mobilnog robota i trajna oštećenja ostalih uređaja, a u nekim slučajevima i opasnost za korisnike mobilnog robota. Uređaje je potrebno i ispravno povezati i integrisati. Greške kod uređaja puno su veće i skuplje od grešaka u kodu.

Uređaji koji se koriste u ovom radu mogu se podijeliti na ulazne (one koji prikupljaju podatke iz okoline), izlazne uređaje (koji šalju obrađene podatke u okolinu) i uređaje za napajanje. Lidar je ulazni uređaj, Arduino, koračni motori i mikrostep drajveri su izlazni uređaji, a napojna jedinica je uređaj za napajanje mobilnog robota.

#### Arduino

**Arduino** razvojna platforma otvorenog koda koja povezuje hardver i softver kako bi se omogućila izrada različitih elektroničkih projekata, od jednostavnih do složenih. To je platforma odlična za povezivanje računara tj. mikrokontrolera sa fizičkim svijetom. Postoji više vrsta Arduina, a u ovom radu se koristi **Arduino Mega 2560**.

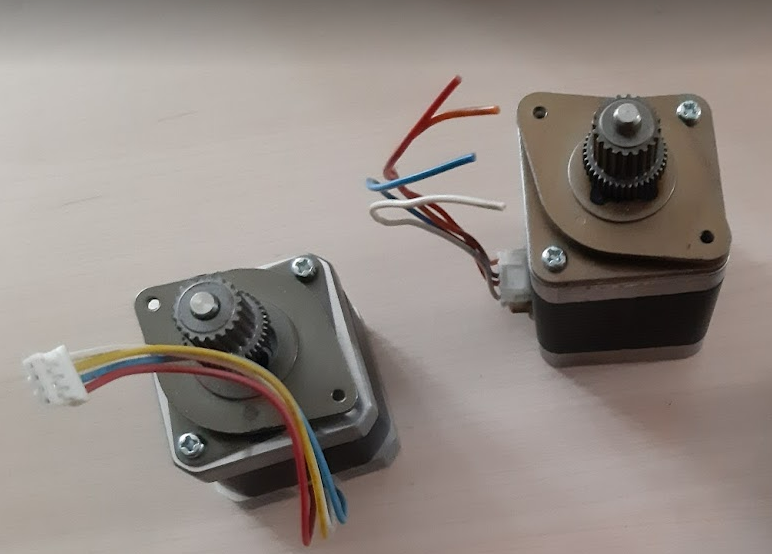
Dizajn Arduino ploče omogućava korisnicima da povezuju senzore poput senzora za svjetlost, vlagu tla, zvuk, protok vode ili mjerenje udaljenosti. Programiranjem ovih senzora mogu se kontrolisati različiti izlazni uređaji, poput motora, dioda, displeja ili releja za AC i DC potrošače. Ove osobine čine Arduino idealnim za razvoj širokog spektra samostalnih projekata.



Slika 3. Arduino MEGA

#### Koračni motori

**Koračni motori** (step motori) su jednosmjerni motori koji se kreću u diskretnim koracima, omogućavajući preciznu kontrolu položaja i brzine. Ovi motori sadrže više zavojnica organiziranih u grupe koje se nazivaju "faze". Aktiviranjem faza u pravilnom redoslijedu motor se rotira korak po korak. Upravljanje koračnim motorom pomoću računara omogućava vrlo precizno pozicioniranje, što ih čini idealnim za aplikacije koje zahtijevaju kontrolu preciznog kretanja. Koračni motori dolaze u različitim veličinama, oblicima i sa različitim električnim karakteristikama, prilagođeni za raznovrsne primjene.

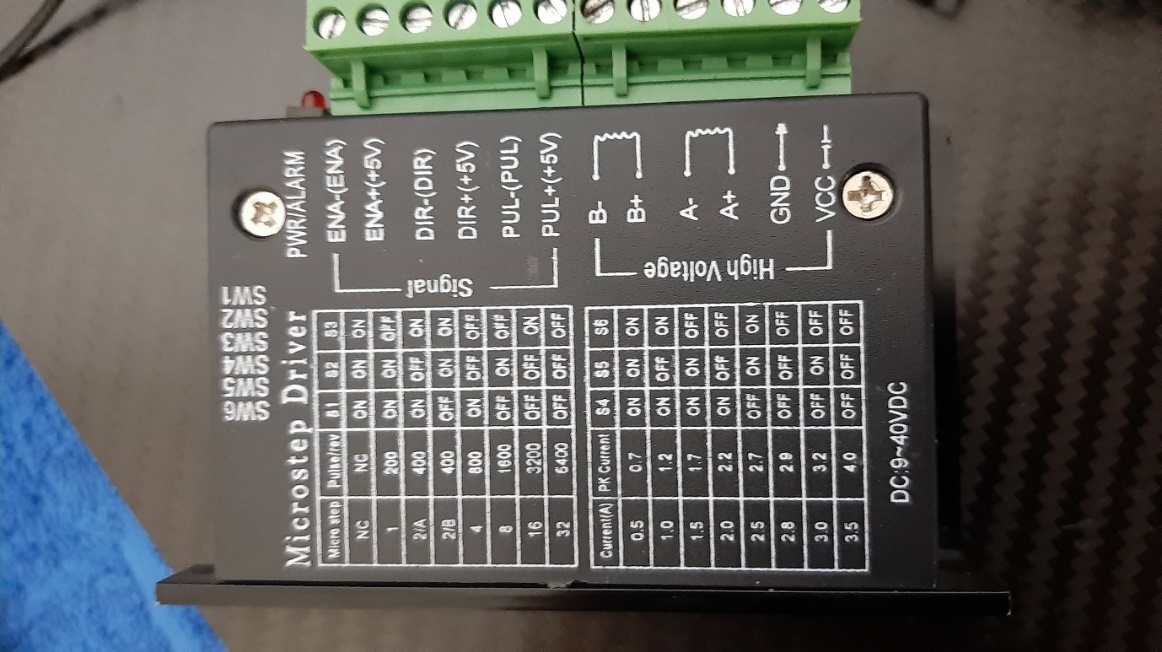


Slika 4. Koračni motori

#### Mikrostep drajveri

**Mikrostep drajveri** su uređaji koji omogućavaju kontrolu pokreta koračnih motora s visokom preciznošću. Oni upravljaju protokom struje prema odgovarajućim namotajima motora kako bi generisali obrtni moment. Specifičnost mikrostep drajvera je ta što dijele struju između faza motora, čime omogućavaju pozicioniranje motora u manjim koracima između standardnih punih koraka. Iako ovo povećava rezoluciju i preciznost kretanja, smanjuje obrtni moment motora.

Na mikorstep drajveru se nalazi 12 pinova, od kojih su prvih 6 **pinovi visokog napona**, a drughih 6 **signalni pinovi**. Od pinova visokog napona **A+** i **A-** kontrolišu jedan set namota (fazu A) motora, **B+** i **B-** kontrolišu drugi set namota (fazu B), i ti pinovi su spojeni sa koračnim motorom. Tu se nalaze još i **VCC** pin na koji se spaja žica sa naponom od 12 V i **GND** pin uzemljenje. Od signalnih pinova **DIR**+ i **DIR** pinovi služe za kontrolu smjera rotacije motora, **PUL**+ i **PUL** pinovi su odgovorni za kontrolu brzine kretanja motora, a **ENA+** i **ENA** su pinovi za enable/disable (omogućavanje/onemogućavanje) motora.

****

Slika 5. Mikrostep drajver

#### Napojna jedinica

**Napojan jedinica (power supply)**, kao što i sam naziv kaže, je glavni izvor napajanja robota. Laptop bi mogao biti glavni izvor napajanja u slučaju da treba vršiti komunikaciju sa LED lampama i manjim senzorima, ali za pokretanje koračnih motora nije dovoljan.

Napojna jedinica je hardverska komponenta koja osigurava napajanje strujom i naponom za računar. Ona omogućava da svaki računalni dio dobije potrebnu količinu energije, budući da različite komponente troše različite količine električne energije. Također, jedan od osnovnih zadataka napojne jedinice je konverzija napona od 220 V na 3,3 V, 5 V i 12 V, čime se zadovoljavaju naponski zahtjevi računalnog hardvera. Napojna jedinica posjeduje vlastiti sistem hlađenja.

****

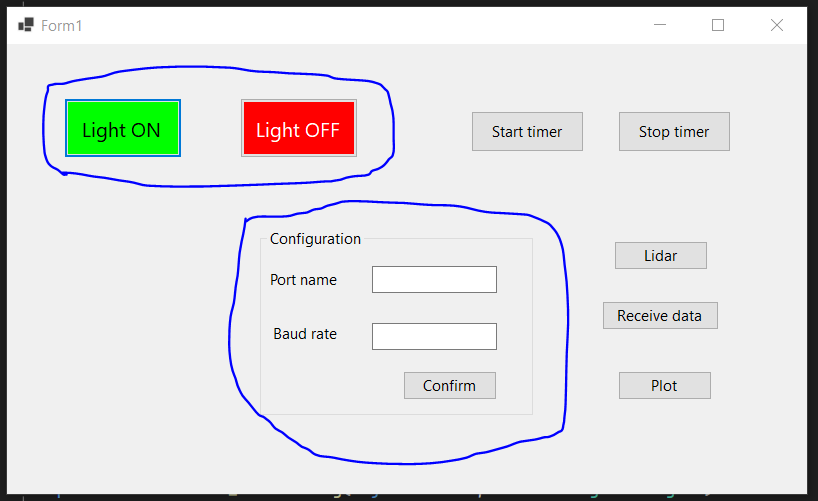
Slika 6. Napojna jedinica

## Pronalazak najkraće putanje

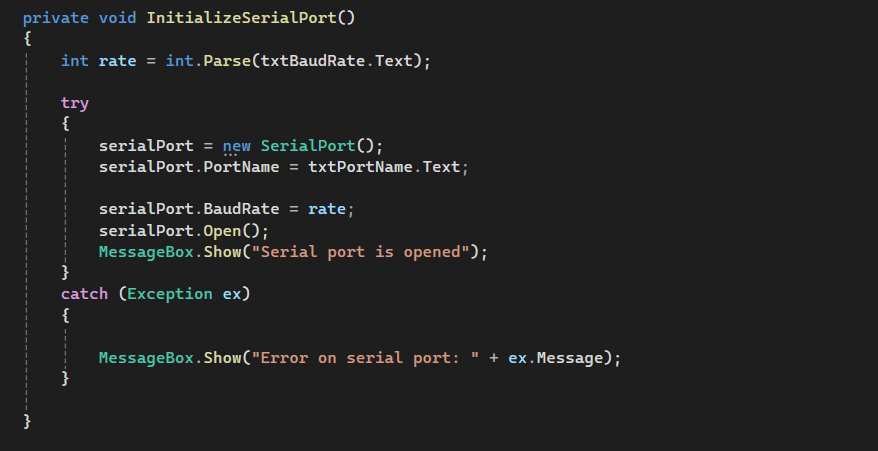
Pronalazak najkraće putanje je bio prvi dio u ovom radu Na tom dijelu se izdvojilo najviše vremena. Taj proces će biti opisan u ovom poglavlju, u kojem se uspostavljala komunikacija sa Lidarom kao ulaznim uređajem i aplikacijom, te dobijeni podaci od Lidara koristili za pronalazak najkraće putanje.

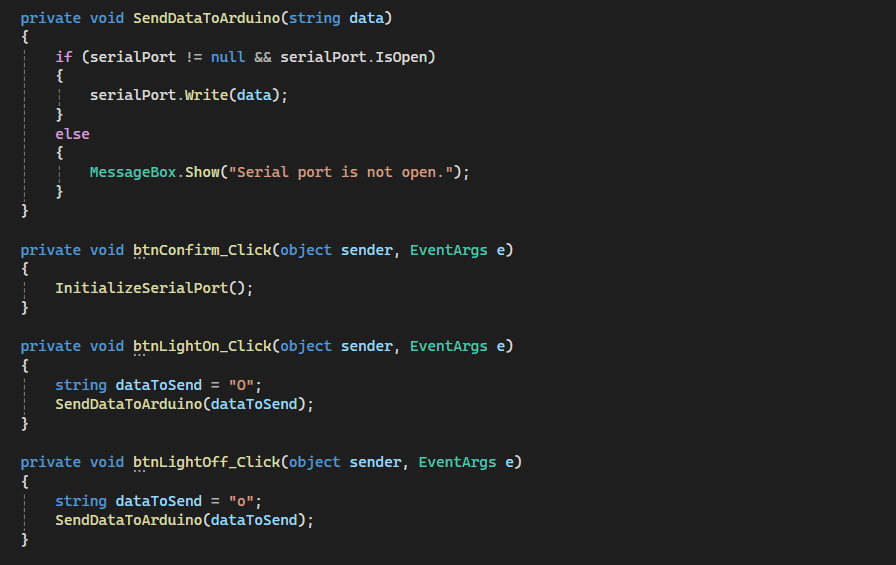
### Uspostava serijeske komunikacije sa Arduinom

Uspostavljena je serijska komunikacija sa Arduinom i C# Windows Forms aplikacijom. Kroz aplikaciju korisnik može paliti i gasiti LED lampu na Arduinu preko grafičkog korisničkog interfejsa. Na slici ispod (zaokruženo plavom bojom) se nalaze dugmad za paljenja i gašenje LED lampe i polja za konfiguraciju serijske komunikacije (unos naziva porta i vrijednost baud rate-a). Također ispod se nalaze slike koda u Windows Forms aplikaciji i Arduino razvojnom okruženju. Kada korisnik pritisne dugme za paljenje ili gašenje, serijskom komunikacijom se šalje poruka na Arduino, a u Arduino razvojnom okruženju je napisan kod koji na osnovu dobijene poruke na LED lampu primjenjuje niski ili visoki napon.



Slika 7. C# Windows Forms aplikacija za paljenje/gašenje LED lampe





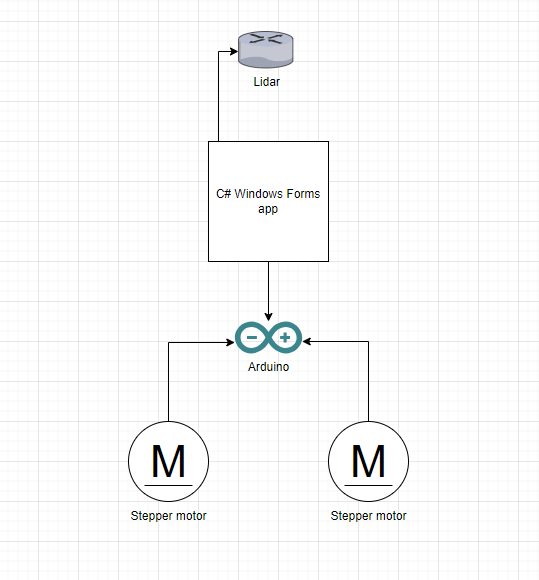
Slika 8. C# kod za paljenje/gašenje LED lampe



Slika 9. Kod u Arduino razvojnom okruženju za paljenje/gašenje LED lampe

### Plan prvog rješenja

Plan prvog rješenja je prikazan na dijagramu na slici ispod. Lidar mapira prostor, šalje podatke u C# Windows Forms aplikaciju koja te podatke pretvara u upotrebljive podatke i izračunava najkraću putanju od početne do ciljne pozicije. C# Windows Forms ima ulogu middleware-a između Lidara i koračnih motora. Zatim se na osnovu putanje šalju komande na Arduno koji upravlja korčanim motorima.

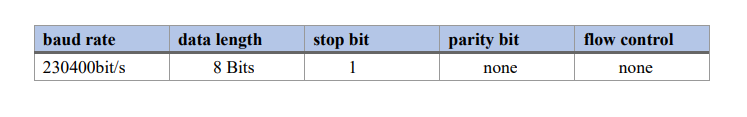


Slika 10. Dijagram prvog plana rješenja mobilnog robota

### Komunikacija Lidara i C# Windows Forms aplikacije

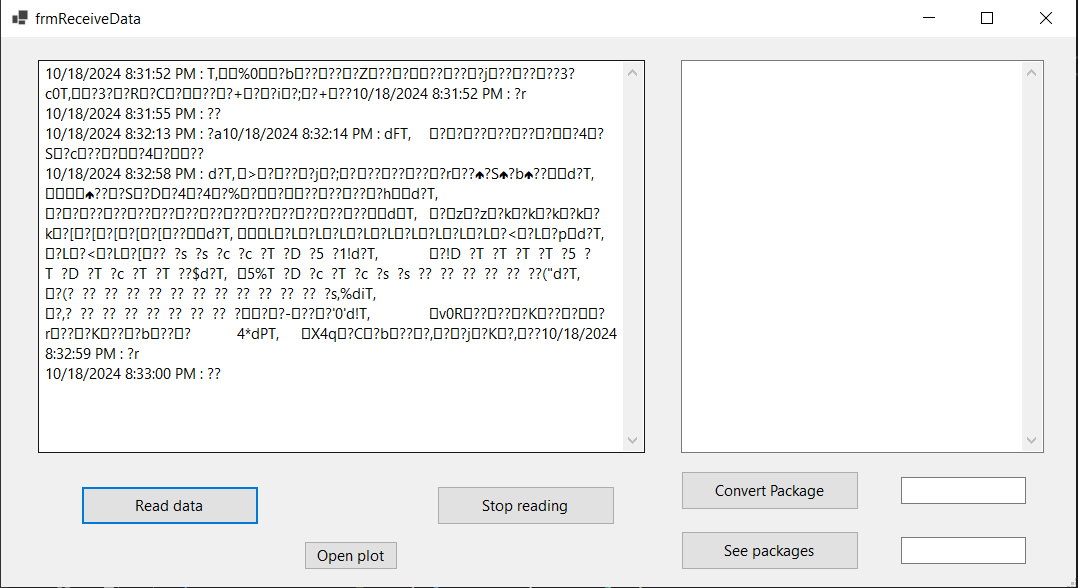
#### Serijska komunikacija sa Lidarom

U C# kodu je ostvarena serijska komunikacija sa Lidarom, slično kao kod Arduina. Razlika je u tome što su se na Arduino slali podaci sa aplikacije, a sada se podaci sa Lidara šalju u aplikaciju. Za ostvarivanje serijske komunikacije potrebno je unijeti naziv porta na koji je Lidar spojen i baud rate 230400. Zatim se kroz event za primanje podataka sa serijskog porta učitavaju podaci u tekstualno polje. Prvobitni podaci dobijeni sa Lidara su stringovi. Njih je prvo potrebno pretvorit u bajtove, a zatim u data pakete.

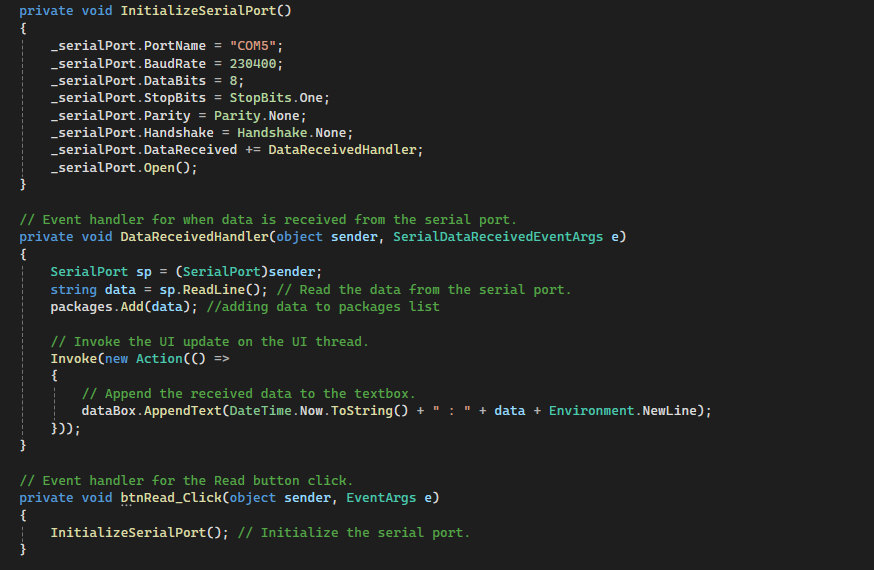


Slika 11. Parametri za transmisiju

Na slici ispod prikazana je forma koja se koristi za prikaz podataka dobijenih od Lidara. Otvaranje serijskog porta i početak ispisa podatak odbijenih od Lidara se vrši pritiskom na dugme „Read data“. Podaci koji su u string formatu se ispisuju lijevom tekstualnom polju.



Slika 12. Prikaz string podataka dobijenih od Lidara u Windows formi





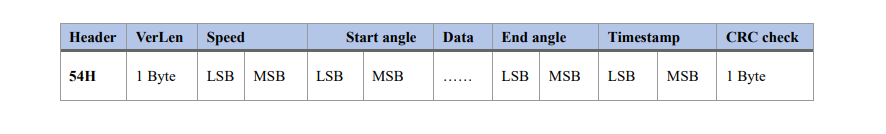
Slika 13. Kod za čitanje podataka sa Lidara

Na slici ispod prikazana je forma koja se koristi za prikaz podataka dobijenih od Lidara. Otvaranje serijskog porta i početak ispisa podataka odbijenih od Lidara se vrši pritiskom na dugme „Read data“. Podaci koji su u string formatu se ispisuju lijevom tekstualnom polju.

#### Pretvaranje podataka u data pakete

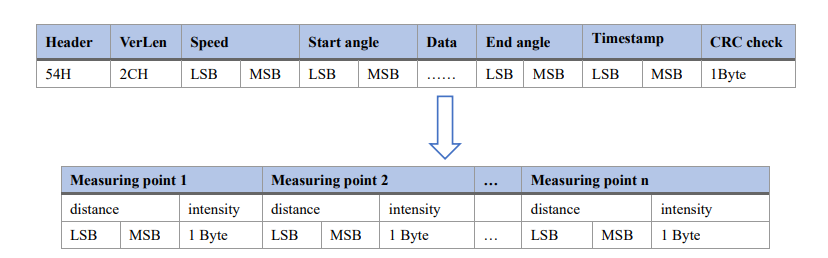
U dokumentaciji o Lidaru objašnjena je struktura data paketa. Data paket se sastoji od slijedećih dijelova:

* **Header** koji je veličine 1 bajt, vrijednost je fiksna na 0x54, što označava početak data paketa
* **VerLen** koji je veličine 1 bajt, gornja tri bita označavaju tip paketa, koji je trenutno fiksiran na 1, a nižih pet bitova označavaju broj mjernih tačaka u paketu, koji je trenutno fiksiran na 12, dakle vrijednost bajta je fiksna na 0x2C
* **Speed** koja je veličine 2 bajta, jedinica je stepeni u sekundi, što označava brzinu Lidara
* **Start angle** koji je veličine 2 bajta, jedinica je 0,01 stepen, što ukazuje početni ugao tačke data paketa
* **Data** označava podatke mjerenja, veličina mjernih podataka je 3 bajta
* **End angle** koji je veličine 2 bajta, a jedinica je 0,01 stepen, što ukazuje na krajnji ugao tačke data paketa
* **Timestamp** koji je veličine 2 bajta, jedinica je milisekunda i maksimum je 30000, a kada dostigne 30000, ponovo će se brojati označavanje vrijednosti vremenske oznake data paketa
* **CRC check** koji je veličine 1 bajt, dobijen je provjerom svih prethodni podataka osim njega samog

****

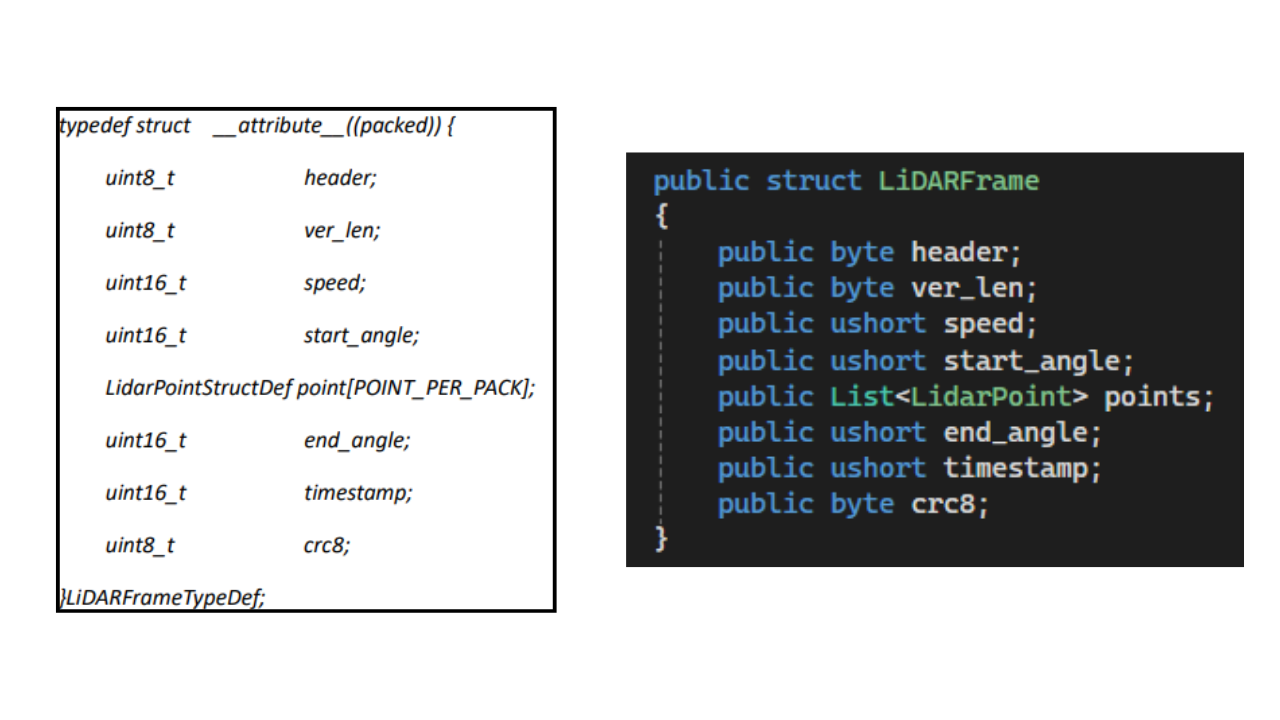
Slika 14. Data paket Lidara

**Data** dio se sastoji od niza od 12 mjernih tačaka. Svaka mjerna tačka se sastoji od 3 bajta, 2 bajta za distancu i 1 bajt za intenzitet signala. Mjerna jedinica za distancu je milimetar.

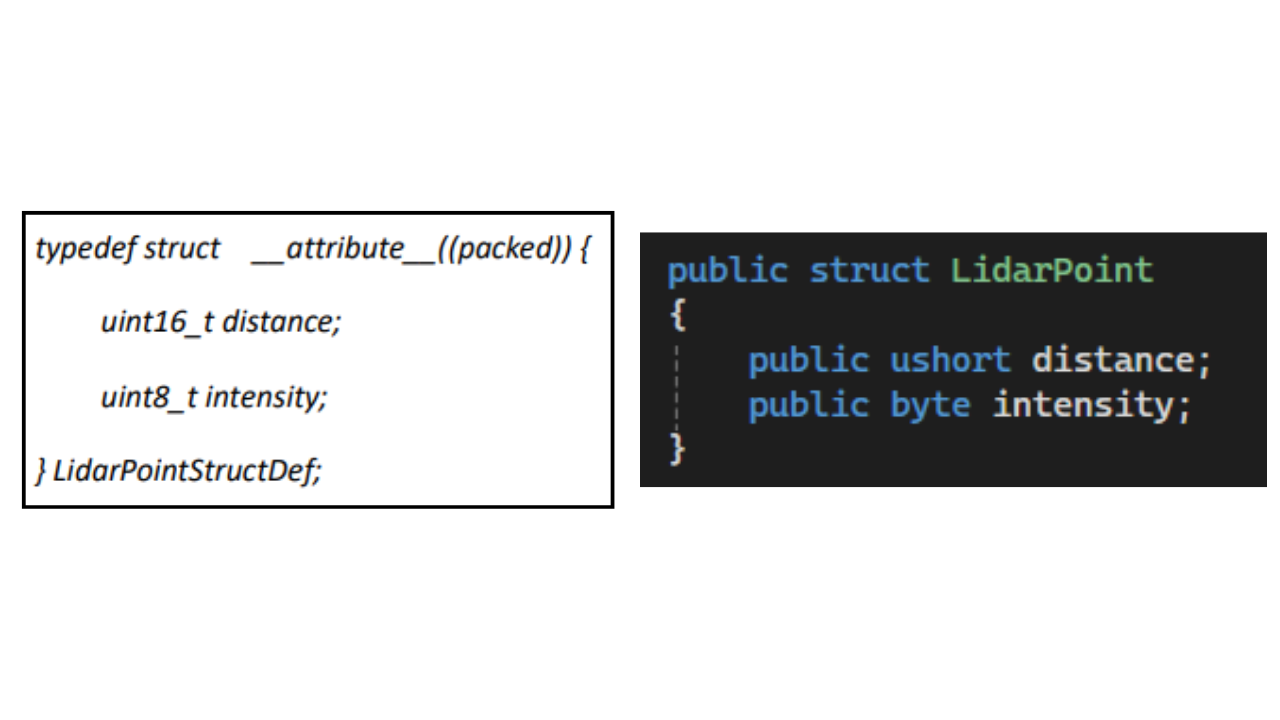


Slika 15. Mjerne tačke Lidara

Po uzoru na struktura u dokumentacija kreirana su strukture u C#-u **LiDARFrame** i **LidarPoint**. LiDARFrame predstavlja data paket, a LidarPoint mjernu tačku.

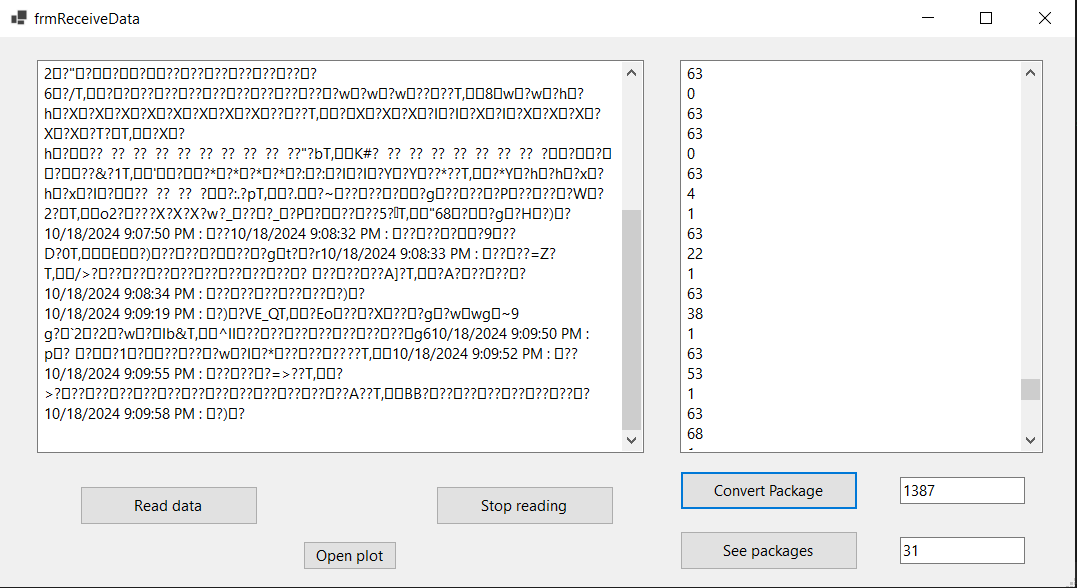


Slika 16. LiDARFrame struktura

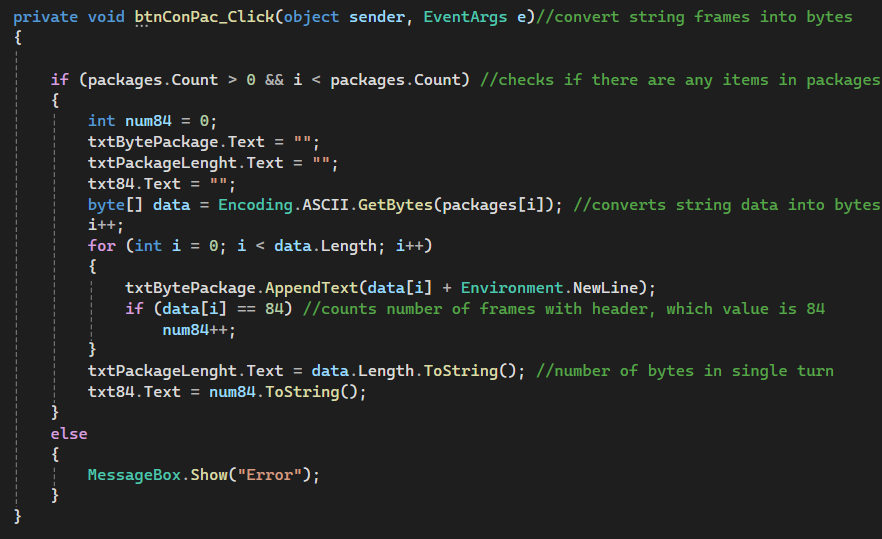


Slika 17. LidarPoint struktura

String podatke prvo je potrebno pretvoriti u bajtove. Prethodno su se nakon svake iteracije dobijenih podataka, podaci spašavali kao string koji se dodavao u string listu “packages”. Na formi korisnik pritiskom na dugme “Convert package” prvi put, prevara prvi string iz string liste „packages“ u bajtove. Svakim narednim pritiskom na pomenuto dugme, pretvara se slijedeći string iz liste. Svaki karakter iz stringa se pretvara u bajt. Karakter u ASCII tabeli ima vrijednost od 0 do 255, a i maksimalna vrijednost bajta je 255, tako da su pretvorene vrijednosti od 0 do 255. U tekstualnim poljima u donje desnom uglu, se nalaze podaci o broju bajtova (gornje tekstualno polje) i broju paketa (donje tekstualno polje) koje se određuje na osnovu broja vrijednosti 84, koja odgovara zaglavlju paketa koji označava početak paketa.



Slika 18. Prikaz podataka u bajtovima na formi (desno tekstualno polje)

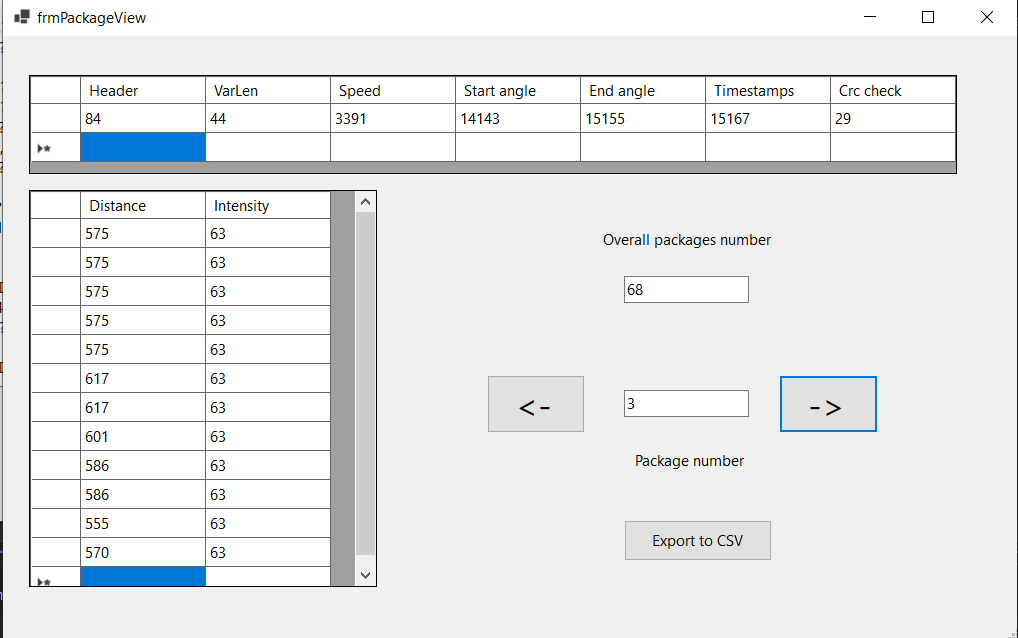


Slika 19. Kod za pretvaranje podataka u bajtove

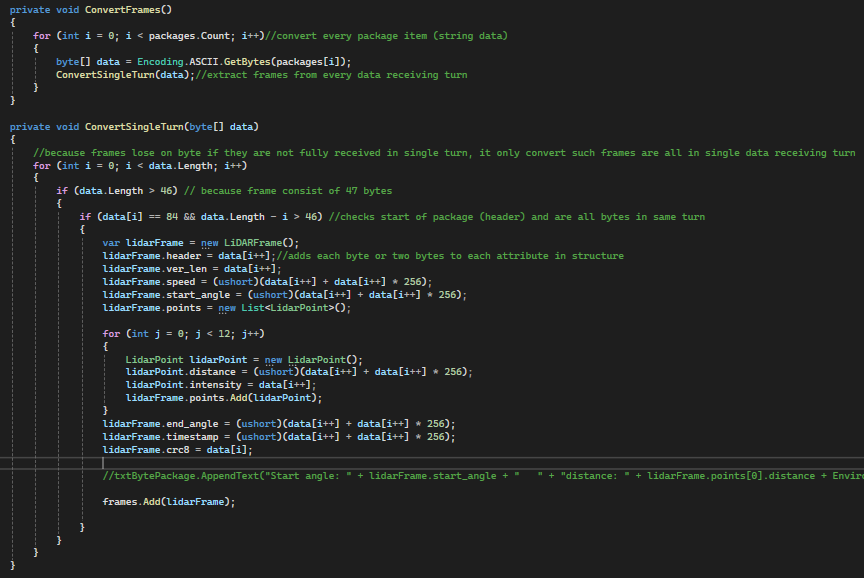
Sljedeći korak je da se bajtovi pretvore u data pakete. Svaka iteracija dobijenih podataka se pretvara zasebno. String se pretvori u niz bajtova, a zatim se niz bajtova proslijedi metodi koja taj niz pretvara u data pakete.

U metodi za pretvaranje niza bajtova u data pakete se prolazi petljom kroz čitav niz bajtova. Prvo se provjerava da li je veličina niza bajtova veća od 46, jer je veličina paketa 47 bajta, a samo potpuni data paketi su validni paketi. Zatim se traži prvi bajt u nizu koji ima vrijednost 84, što predstavlja zaglavlje koje označava početak paketa. Kada se pronađe takav bajt, potrebno je provjeriti i da li se od tog elementa u nizu bajtova pa do kraja nalazi barem još 47 elemenata, kao bi se kreirao cjelovit data paket. Data paketi se ne mogu pretvoriti ako se njegovi dijelovi nalaze u više iteracija, jer se u svakoj iteraciji gubi po jedan bajt, a nepotpuni data paketi nisu validni. Ako su ti uslovi tačni inicijaliziraju se svi atributi strukture LiDARFrame, među kojim je i lista tačaka. Ako je atribut veličine dva bajta, trenutni i naredni bajt se pretvaraju u 16-bitni cjelobrojni nenegativni tip podataka (engl. 16-bit unsigned integer) **ushort**. U jednoj iteraciji moguće je kreirati više data paketa.

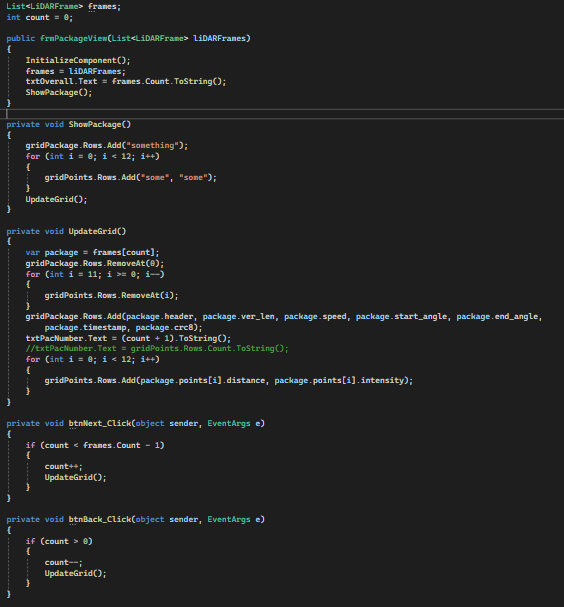
Na formi za prikaz podataka dobijenih od Lidara se nalazi dugme „See packages“. Kada se to dugme pritisne, izvrši se kod za pretvaranje podataka u data pakete, otvori se nova forma za pregled paketa na koju se pošalju pretvoreni data paketi. Na toj formi su prikazani svi dijelovi data paketa i njihove vrijednosti, kao i distanca i intenzitet svih 12 tačaka u paketu. Na formi je moguće kretati se po listi data paketa, naprijed i nazad.



Slika 20. Forma za prikaz data paketa



Slika 21. Kod za pretvaranje podataka u data pakete



Slika 22. Kod za prikaz data paketa u formi

### Nedostaci prvog rješenja i novo rješenje

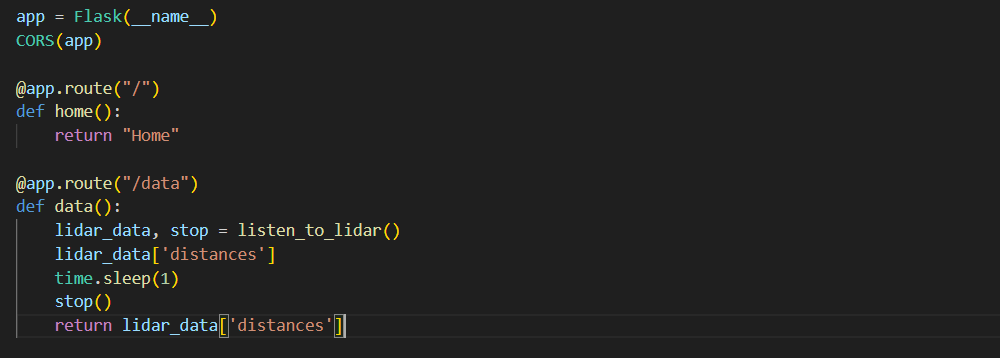
Pronađen je novi GitHub profil koji je referenciran na GitHub profilu od Lidara. Tu se nalazio kod za komunikaciju sa Lidarom u Python programskom jeziku. U tom repozitoriju se nalze 2 Python fajla, jedan koji sluša podatke sa Lidara (listen\_to\_lidar.py) i drugi koji računa i preoblikuje podatke sa Lidira (calc\_lidar\_data.py). Tada se promijenio prvobitni plan u kojem je pored C# Windows Froms aplikacije uključen i Python program. Ideja je bila da se u Python programu podaci sa Lidara pretvore u listu tačaka sa informacijama o uglu i distanci prepreka, i da se te tačke pošalju na C# Windows Forms aplikaciju preko API-a.

****

Slika 23. Kod u Python programu za kalkulaciju podataka dobijenih od Lidara

#### Komunikacija Lidar, Python programa i C# Windows Forms aplikacije

U Python programu napravljen je API korištenje modula **Flask**. Endpoint „data“ vraća dictionary u kojem je key vrijednost ugla a value vriejdnost distance.



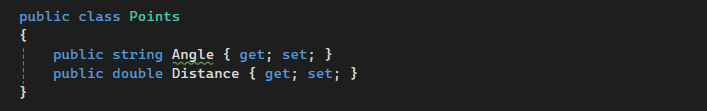
Slika 24. API poziv za dohvatanje tačaka

U C# Windows Froms aplikaciji se poziva API endpoint. Potrebna je metoda koja će pozivati API endpoint tako što joj se pošalje URL, a ona vraća rezultat tog poziva. Na formi se treba nalaziti dugme sa kojim će se pozivati API poziv. Kada se podaci dohvate potrebno ih je spasiti u dictionary u C#. Zatim sve tačke koje su spašene u dictionary „dots” je potrebno iscrtati, a to se postiže pomoću **Paint** eventa u formi.

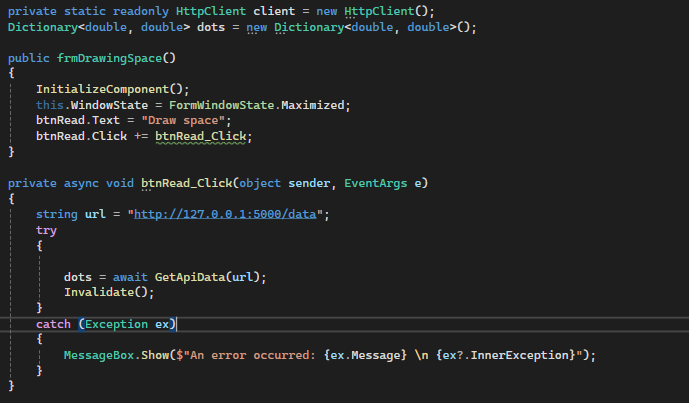
A screen shot of a computer code

Description automatically generated

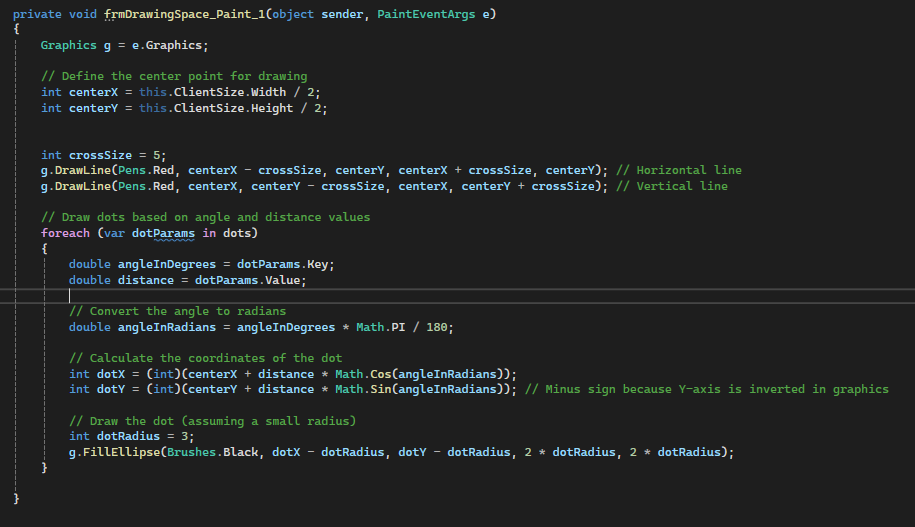
Slika 25. Metoda u C#-u za API poziv



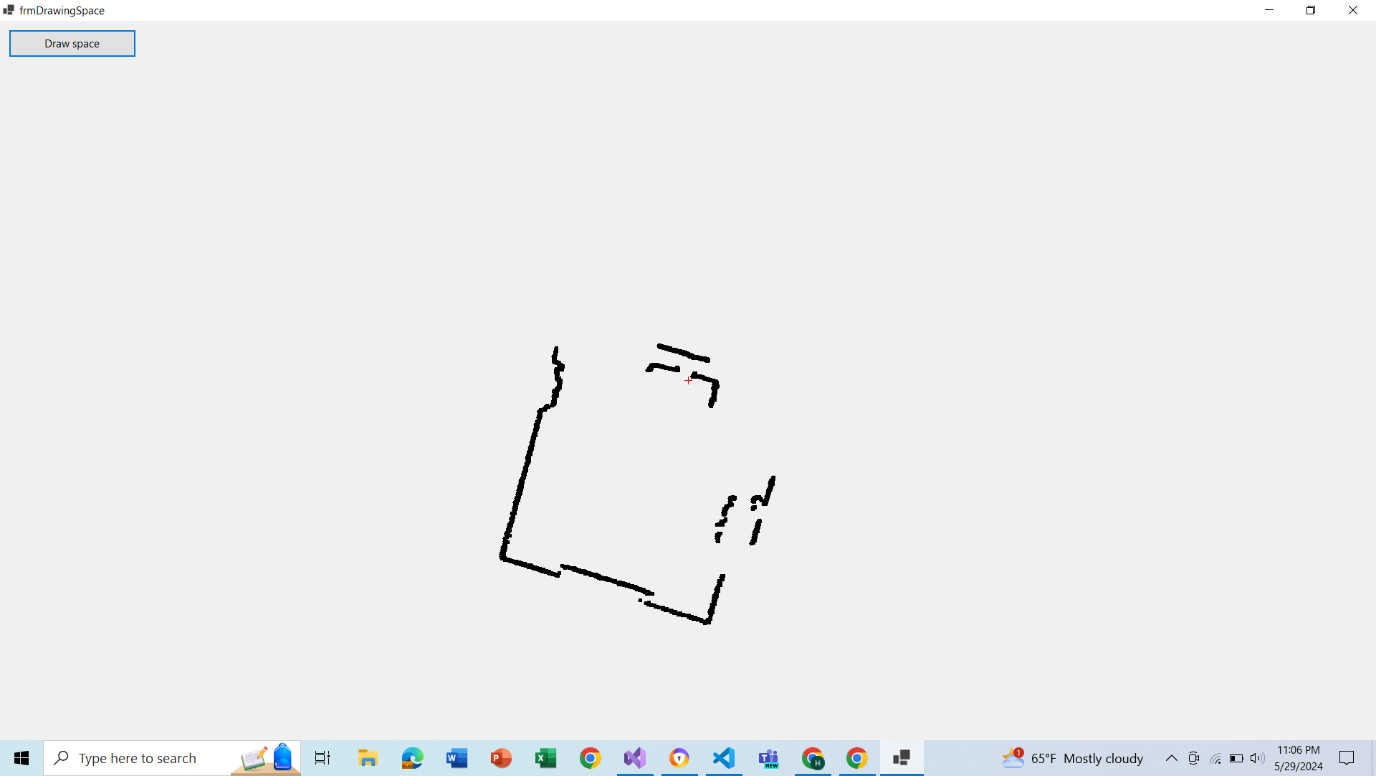
Slika 26. Klasa u C#-u za tačku koja sadrži ugao i distancu

****

Slika 27. Forma i dugme za poziv API-a

****

Slika 28. Event za iscrtavanje tačaka

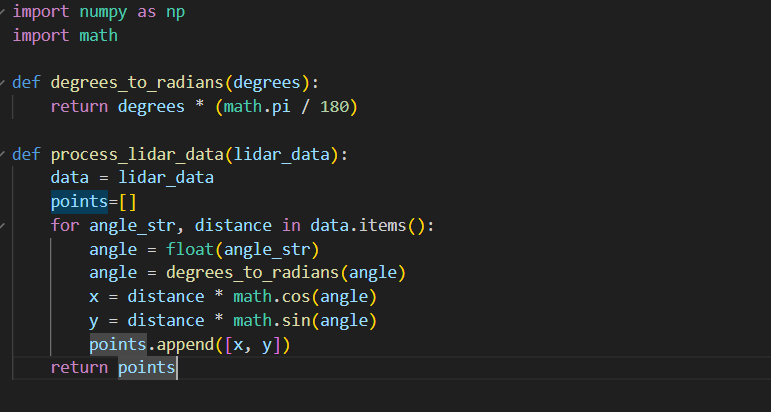


Slika 29. Iscrtani prostor u C# aplikaciji

#### Prikaz mapiranog prostora u Python programu

U Pythonu je također moguće iscrtati tačke pomoću modula **matplotlib**. Pošto je i obradu podataka i prikaz mapiranog prostora moguće izvršiti u Python programu odustaje se od C# aplikacije, jer potreba za komunikacijom dvije aplikacije dosta otežava čitav proces, što bi se moglo odraziti u nastavku rada.

Dobijene tačke sa podacima u uglu i distanci potrebno je pretvoriti u kartezijanske tačke, tj. tačke u koordinatnom sistemu sa vrijednostima x i y. Vrijednosti uglova koje su u stepenima potrebno je pretvoriti u radijane, a zatim izračunati x i y vrijednost za svaku tačku. X se dobije tako što se distanca pomnoži sa kosinusom ugla u radijanima, a y kada se distanca pomnoži sa sinusom ugla. Funkcija za pretvaranje u kartezijanske tačke se dodaje u zaseban fajl „functions“, u kojem će se nalaziti sve funkcije potrebne za manipulaciju nad podacima dobijenih od Lidara.



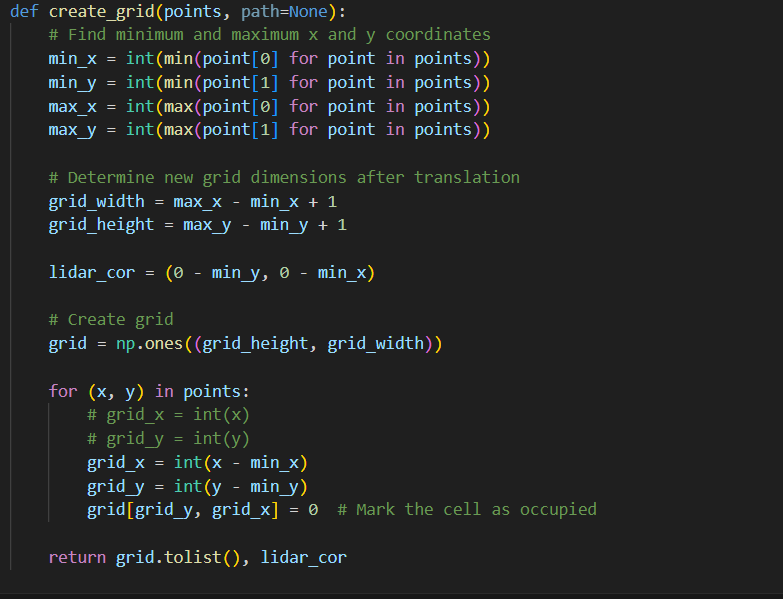
Slika 30. Metode za pretvaranje u kartezijanske tačke

#### Priprema podataka za algoritam pretrage

Nakon što je je kumunikacija između Lidara i Python programa uspješno uspostavljena, te prostor uspješno mapiran, potrebno je pripremiti ulazne podatke za algoritam pretrage. Algoritam će pronalaziti najkraću putanju u dvodimenzionalnom gridu, što u kodu predstavlja dvodimenzionalni niz sa nulama i jedinicama. Jedinice predstavljaju slobodan prostor, a nule prepreke. Tačke iscrtanog prostora predstavljaju prepreke. Indeksi reda i kolone u gridu prestavljaju koordinate u prostoru. Dakle ako tačka ima koordinate (23,74), element u gridu koji se nalazi u 74. redu i 23. kloni će imati vrijednost 0. Niz će na početku biti inicijaliziran sa svim jedinicama, a onda će svim elementima koji odgovaraju tačkama prepreka dodijeliti vrijednost 0.

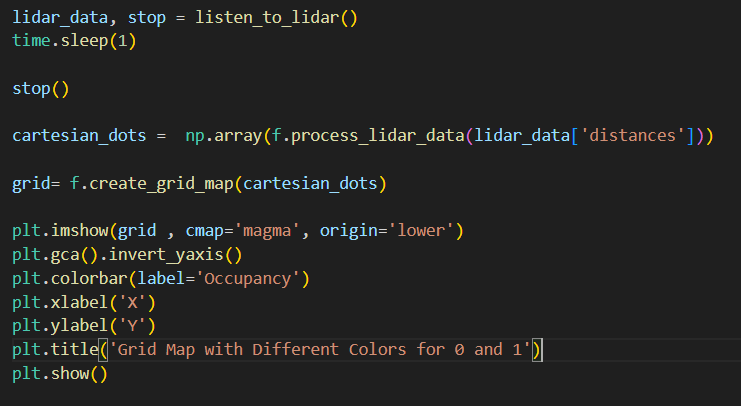
Za kreiranje grida će biti kreirana metoda koja prima kartezijanske tačke, a koja kao rezultat vraća dvodimenzionalni niz koji odgovara gridu maprianog prostora. Međutim nije dovoljno samo proslijediti kartezijanske tačke i na osnovu koordinata tačaka dodijeliti nule elementima niza koji odgovaraju tačkama. Niz ne može imati negativne indekse, a tačke u koordinatnom sistemu mogu. U kartezijanskim tačkama će vjerovatno biti negativnih indeksa, što nije prihvatljivo. Da bi se taj problem riješio, prvo je potrebno pronaći najmanje vrijednosti koordinata na x i y osi, a zatim od svake koordinate na x osi oduzeti najmanju vrijednost x koordinate i od svake koordinate na y osi oduzeti najmanju vrijednosti y koordinate. To se naziva prevođenje koordinata, i sada će najmanja vrijednost koordinata x i y ose biti 0.

Također metoda vraća i koordinate na kojima se nalazi Lidar. Prije prevođenja koordinata, koordinate Lidara su bile (0,0), jer je Lidar u centru koordinatinog sistema, a distance prepreka su zapravo udaljenosti od Lidara. Za prevođenje koordinata pozicije Lidara vrijedi isti postupak kao i za sve ostale tačke.

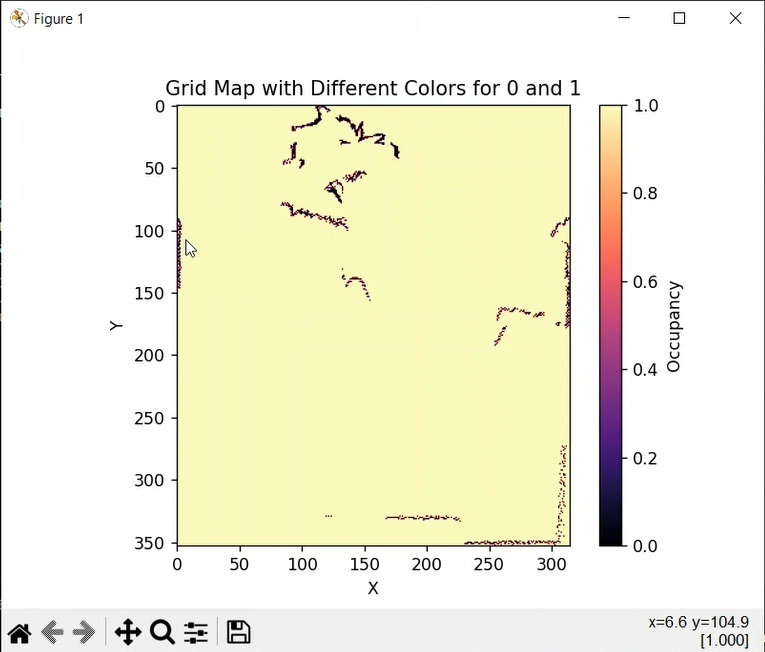


Slika 31. Metode za kreiranje grida

Nakon kreiranja metode za kreiranje grida, prostor koji mapira Lidar moguće je prikazati u Python programu. Podaci o mapiranom prostoru se učitaju u program kao dictionary čiji je key ugao a value distanca. Taj dictionary se prosijedi u metodu koja ga pretvara u kartezijanske tačke. Zatim se kartezijanske tačke prosljeđuju u metodu koja kreira grid. Kreirani grid se kroz matplotlib modul vizuelno prikazuje.



Slika 32. Proces vizualizacije mapiranog prostora



Slika 33. Vizualizirani grid (mapirani prostor)

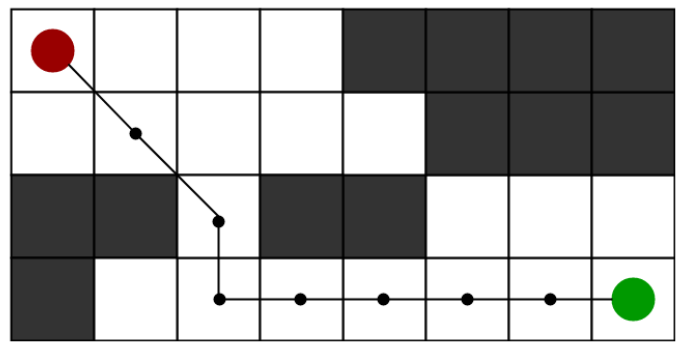
### Algoritam pretrage

Nakon što je grid kreiran, svi ulazni podaci su spremni. Slijedeći korak je odabrati algoritam pretrage koji će se koristiti za pronalazak najkraće putanje od početne do ciljne pozicije. Python program treba korisniku da omogućiti odabir ciljne pozicije.

#### A\* algoritam pretrage

A\* algoritam pretrage se korsiti za pronalazak putanje i pretragu grafa, i jedan je od najboljih tehnika za to. Ovaj algoritam pretrage se razlikuju od ostalih tehnika pretraživanje jer je „inteligentan“.

A\* algoritam je najlakše objasniti u slučaju kada u kvadratnom gridu imamo označenu početnu i ciljnu poziciju. U tom slučaju želimo doći do cilja najkraćim putem a pri tome izbjegavati prepreke. Algoritam vrši provjere nakon svakog pomaka uzimajući u obzir dva parametra. Prvi parametar **g** prestavlja trošak kretanja od početne do date pozicije u gridu, uzimajući obzir putanju koja vodi do date pozicije. Drugi parametar **h** prestavlja trošak kretanja od date do ciljne pozicije. Parametar h se odnosi na heuristiku. Heuristika predstavlja pametno pretpostavljanje putanje, jer stvaran dužina putanje dok se ne pronađe je nepoznata zato što se na putu mogu naći prepreke. To je glavna razlika između Dijkstra algoritma i A\* algoritma, jer Dijkstra algoritam ne koristi heuristiku.



Slika 34. Primjer korištenja A\* algoritma pretrage

U kodu algoritma se kreiraju dvije liste čvorova, otvorena lista i lista neposjećenih čvorova. Kod algoritma funkcioniše na slijedeći način:

1. **Inicijalizacija otvorene liste čvorova:** Dodaje se početni čvor, koji je početna tačka pretrage.
2. **Inicijalizacija liste posjećenih čvorova:** Na početku je prazna.
3. **Glavna petlja pretrage:**

* Dok god otvoren lista nije prazna, algoritam ponavlja sljedeće korake:

1. Pronalazi čvor s najmanjom ukupnom vrijednošću ocjene **f** u otvorenoj listi, nazvan „**q**“.
2. Uklanja čvor „q“ s otvorene liste i dodaje ga u listu posjećenih čvorova.
3. **Generisanje sukcesora (nasljednika):**

* Generišu se nasljednici čvora „q“ (najčešće se uzima 8 okolnih tačaka ako je mreža dvodimenzionalna) i postavlja „q“ kao roditeljski čvor svakog nasljednika.

1. **Računanje ocjena za svakog nasljednika**:

* Za svakog nasljednika se provjerava:
  + Ako je cilj, zaustavlja se pretraga
  + Ako nije, izračunava se:
    - **g** vrijednost (udaljenost od početnog čvora do trenutnog nasljednika),
    - **h** vrijednost (heuristička procjena udaljenosti do cilja, koja može koristiti Manhattan, Dijagonalnu ili Euklidsku heuristiku),
    - **f** vrijednost (g + h).
  + Ako postoji čvor u otvorenoj listi sa istom pozicijom i nižom vrijednošću f, preskače se ovaj nasljednik.
  + Ako postoji čvor u listi posjećenih čvrova s nižom vrijednošću f, također se preskače nasljednik. U suprotnom, dodaje se nasljednik u otvorenu listu.

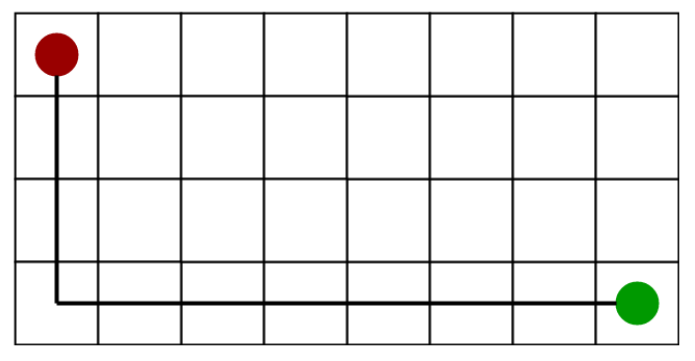
1. **Dodavanje čvora u listu posjećenih čvorova:**

* Čvor „q“ se dodaje u listu posjećenih čvorovakako bi se spriječilo vraćanje na njega tokom pretrage.

**Manhattan udaljenost** predstavlja zbir apsolutnih vrijednosti razlike između x i y koordinata ciljne tačke i trenutne ćelije. Na ovaj način, računanje udaljenosti funkcioniše po formuli:

* h = abs(current\_cell.x - goal.x) + abs(current\_cell.y - goal.y)

Ova metoda omogućava procjenu broja poteza potrebnih da se dođe do cilja kada su dozvoljeni samo horizontalni i vertikalni potezi, što je korisno u primjenama poput A\* algoritma.

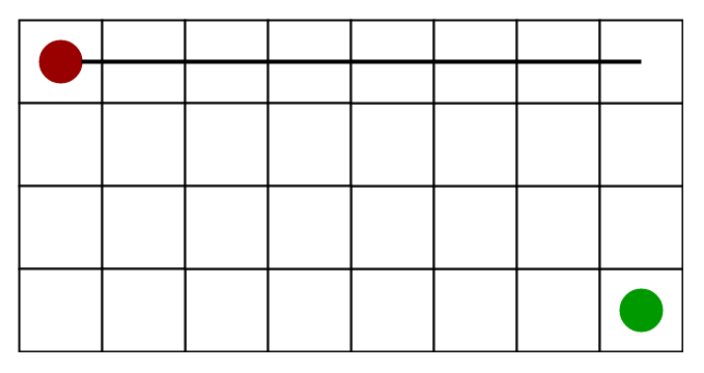


Slika 35. Manhattan udaljenost

**Dijagonalna udaljenost** između trenutne tačke i ciljne tačke definiše se kao maksimum apsolutnih vrijednosti razlika između njihovih x i y koordinata. Preciznije, prvo računamo apsolutne razlike dx i dy između x i y koordinata trenutne tačke i cilja. Zatim se udaljenost h određuje formulom:

* udaljenost = D × (dx + dy) + (D₂ - 2 × D) × min(dx, dy),

gdje D predstavlja dužinu svakog čvora (obično 1), a D₂​ je dijagonalna udaljenost između čvorova (obično √2). Ova heuristika je korisna kada je dozvoljeno kretanje u osam smjerova, slično kretanju kralja u šahu.

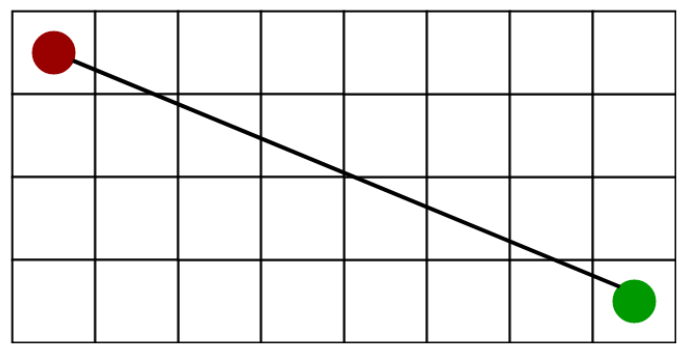


Slika 36. Dijagonalna udaljenost

**Euclidova udaljenost** predstavlja udaljenost između trenutne tačke i ciljne tačke koristeći formulu za izračunavanje udaljenosti. Ona se izražava kao:

* h = sqrt ( (current\_cell.x – goal.x)2 + (current\_cell.y – goal.y)2 )

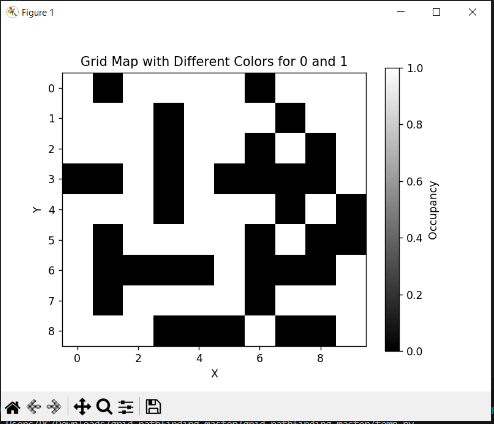
Ova heuristika se koristi kada je dozvoljeno kretanje u svim pravcima, bez ograničenja. U ovom kontekstu, Euclidova udaljenost omogućava precizno izračunavanje rute do cilja, što može biti korisno u aplikacijama gdje je važno kretanje u bilo kojem smjeru.



Slika 37. Euclidova udaljenost

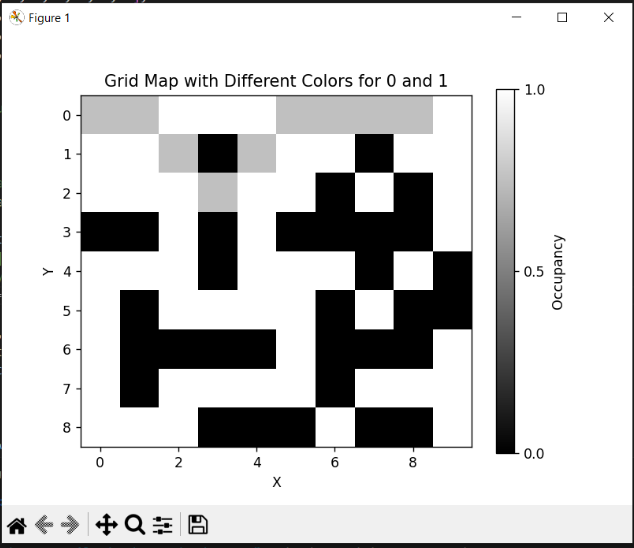
#### Testiranje algoritma

Objašnjenje i kod za A\* algoritam pronađeni su na web stranici **GeeksforGeeks**. Taj kod omogućuje kretanje u svim pravcima, uključujući i dijagonalna kretanja. Međutim prije upotrebe koda na mobilnom robotu, potrebno je testirati algoritam. Za testiranje je korišten manji grid. Na slici ispod se nalazi grid koji je korišten za testiranje. Crna polja predstavljaju prepreke, bijela slobodan prostor, a siva izračunatu najkraću putanju od početka do cilja.



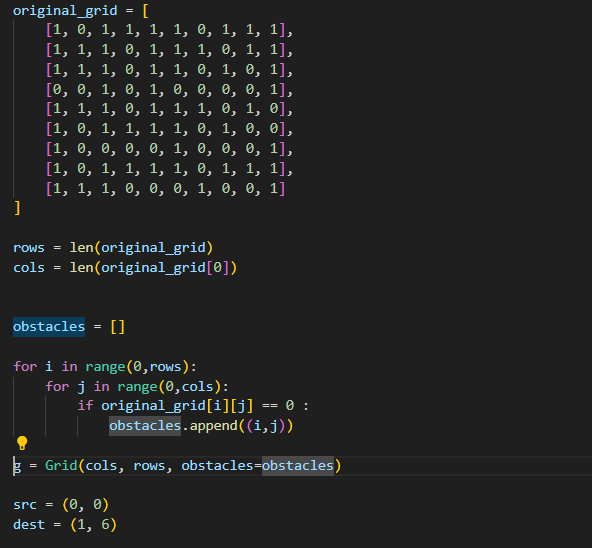
Slika 38. Prikaz testnog grida

Na slici ispod sivom bojom su označena polja putanje koju je generisao A\* algoritam. U poređenju sa prvom slikom, vidi se da se putanja ne kreće samo po slobodnom prostoru, već i da određena crna polja zamijenjena sivom bojom. To znači da putanja ne zaobilazi prepreke veći ide preko njih što nije ispravno. Zbog toga ovaj kod A\* algoritma pretrage nije validan i potrebno pronaći drugi kod.

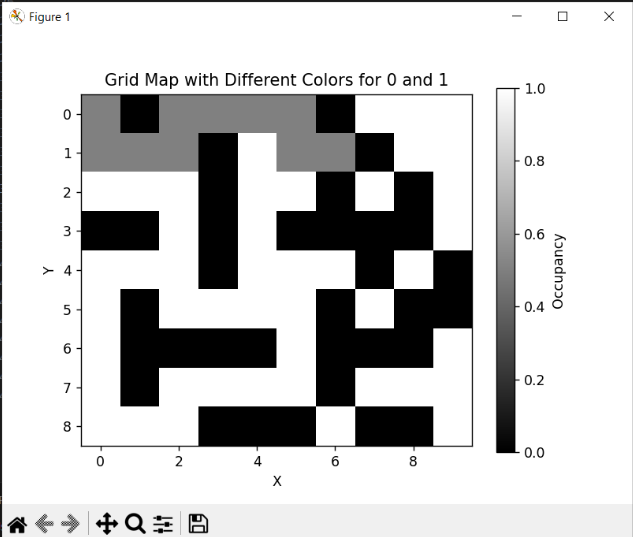


Slika 39. Prikaz generisane putanje prvim kodom A\* algoritma pretrage

Na GitHub profilu **Ming Yean Lim**, u repozitoriju **grid-pathfinding**, pronađen je alternativni kod za A\* algoritam pretrage. Također, u ovom repozitoriju nalazi se korisnički definisana klasa **Grid**, koja omogućava lakše upravljanje u poređenju s primitivnim dvodimenzionalnim nizom. Konstruktor klase Grid zahtijeva unos tri parametra: broj redova, broj kolona i listu prepreka, čiji su elementi **tuple** s indeksima dvodimenzionalnog niza koji predstavljaju koordinate prepreka. Ovaj kod algoritma je uspješno pronašao najkraću putanju od početne do ciljne pozicije izbjegavajući prepreke. Ali za razliku od prvog koda, sa ovim algoritmom nije moguće kretati se dijagonalno.



Slika 40. Kod za testiranje drugog A\* algoritma pretrage

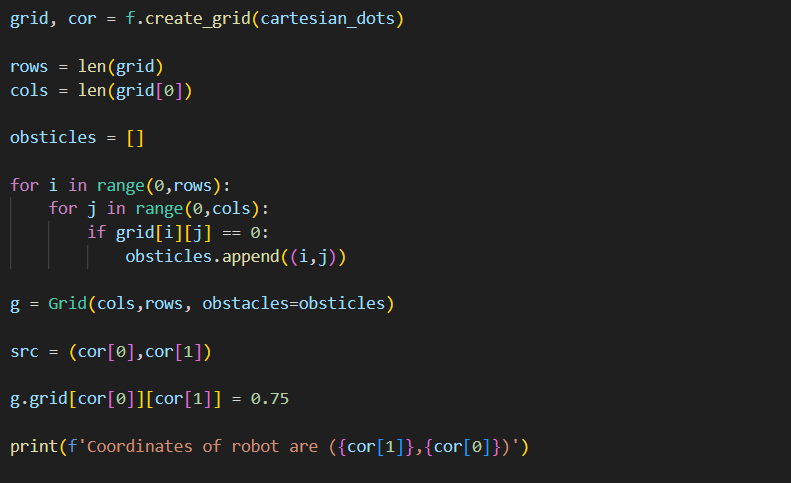


Slika 41. Pikaz uspješnog pronalaska najkraće putnaje na testnom gridu

#### Primjena algoritma

Nakon što je algoritam uspješno testiran, može se koristiti za pronalazak najkraće putanje u prostoru koji je mapirao Lidar. Kod za računanje najkraće putanje je zapravo nastavak koda za vizualizaciju grida (slika 31.)

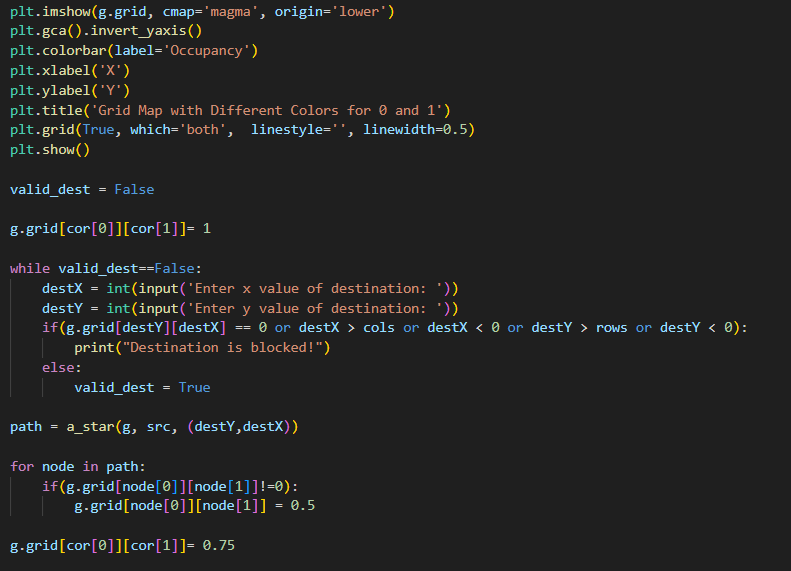
Funkcija za kreiranje grida vraća i koordinate pozicije Lidara. Pozicija Lidara će biti početna pozicija u algoritmu. Poslije toga je potrebno pripremiti parametre za klasu Gird. Izračunati su brojevi redova i kolona ,a na osnovu dvodimenzionalnog niza koji prestavlja mapirani prostor, pronađene su koordinate prepreka i dodane u niz. Taj niz je treći parametar klase Grid. Kreira se instanca klase Grid, a elementu koji odgovara poziciji Lidara se dodjeljuje vrijednost 0.75, kako bi ta pozicija bila predstavljena drugačijom bojom.



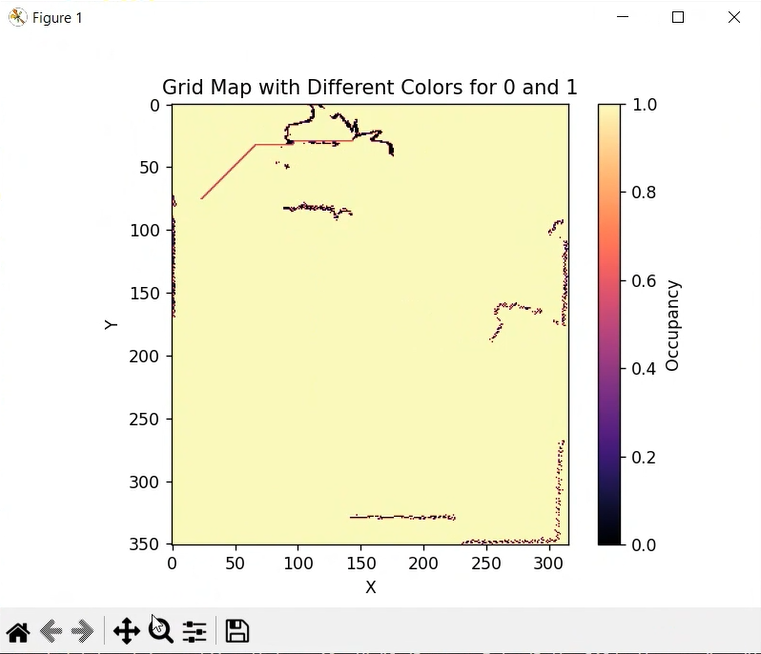
Slika 42. Kreiranje instance klase Grid

Nakon deklarisanja i instanciranja klase Grid, mapirani prostor i pozicija Lidara se vizuelno prikazuju. Na vizuelnom prikazu korisnik može odrediti željenu destinaciju. Nakon zatvaranja vizuelnog prikaza, korisnik unosi koordinatu ciljne pozicije, a u kodu se vrši provjera da li se koordinata nalazi unutar dimenzije prostora i da li je pozicija destinacije slobodan prostor, jer nije dozvoljeno kretanje po preprekama.

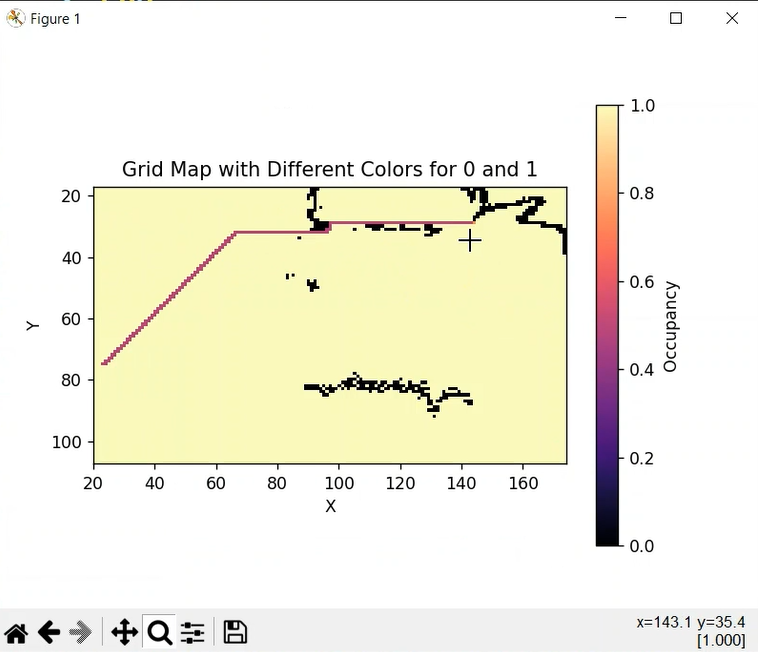
Ako je koordinata ispravna, poziva se funkcija koja upotrebom A\* algoritma pretrage pronalazi najkraću putanju. U funkciju se unose parametri: grid, početna i ciljna pozicija . Nakon što se izračunata putanja pohrani u varijablu, potrebno je svim koordinatama, koje odgovaraju putanji, dodijeliti vrijednost 0.5, kao bi putanja bio prikazan drugačijom bojom. I na kraju, potrebno je opet prikazati prostor sa najkraćom putanjom od početne do ciljne pozicije.



Slika 43. Kod za unos ciljne pozicije i računanje najkraće putanje



Slika 44. Vizualni prikaz izračunate najkraće putnje



Slika 45. Uvećani prikaz kako algoritam uspješno izbjegaje prepreke

## Kreiranje izvšnog dijela mobilnog robota

Nakon pronalaska najkraće putanja završen je dio mobilnog robota koji dohvata i obrađuje podatke. Drugi dio mobilnog robota je izvršni dio. Nakon što Python program obradi podatke, komade se šalju izvršnom fizičkom dijelu koji pokreće mobilni robot.

### Sklapanje i povezivanje fizičkih komponenti

Posložiti sve fizičke komponente (Lidar, laptop, napojna jedinica, koračni motori, mikropstep drajveri i Arduino) na malom prostoru koji se kreće i povezati ih žicama je bilo jako izazovno. Kao platforma, na koji će se nalaziti svi dijelovi, odabrana je ploča od karbonskih vlakana na aluminijskoj konstrukciji, površine 50 x 40 cm2 . Na ploču su pričvršćena dva točak koja su 3D printani. U točkove su ugrađeni koračni motori. Na drugom kraju platforme silikonom je zalijepljen okretni točak. Taj točak nema pogona, već služi kao pomoćni točak.

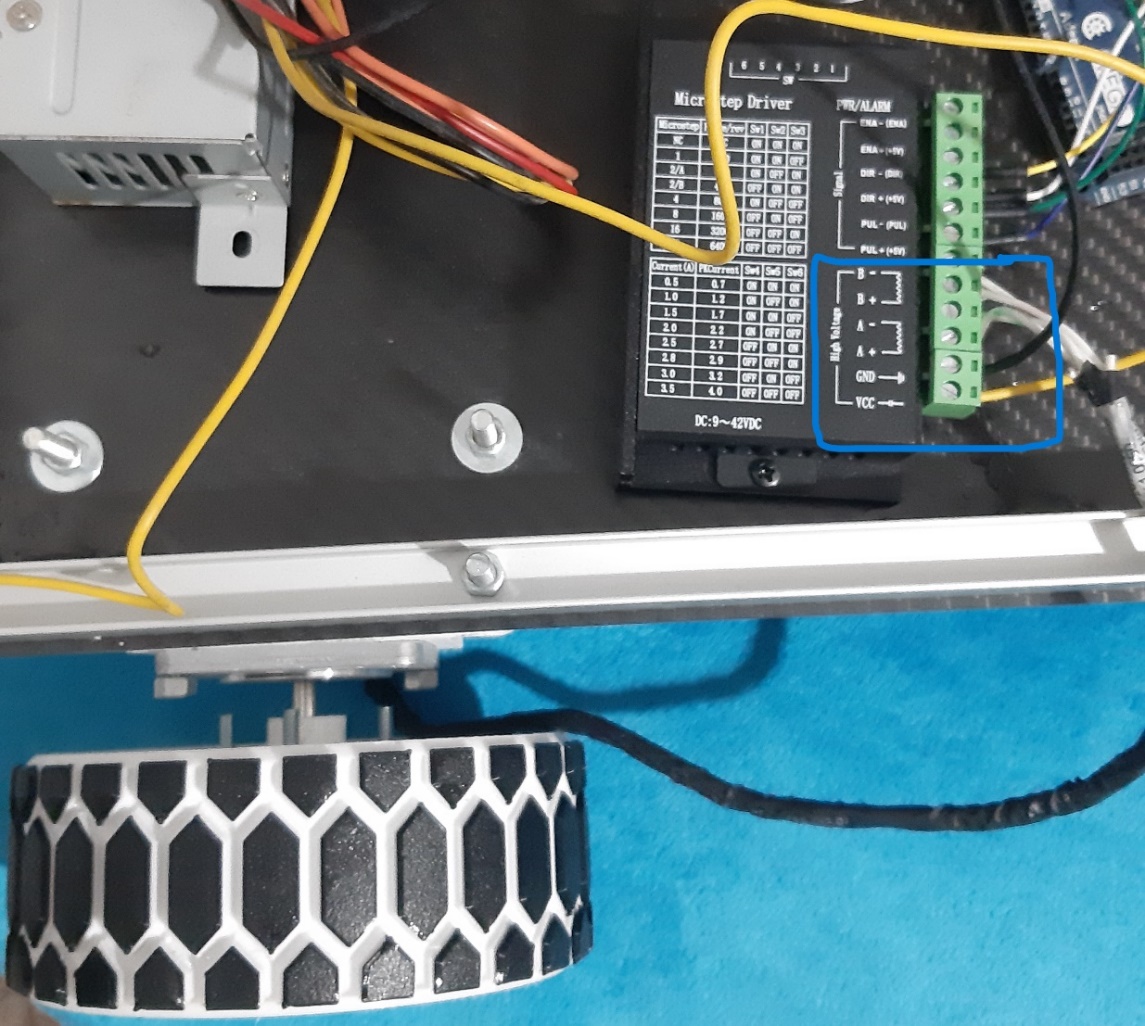


Slika 46. Platforma mobilnog robota

#### Spajanje koračnih motora sa mikrostep drajverima

Koračne motore potrebno je spojiti sa mikrostep drajverima. Mikorstep drajveri imaju veoma važnu ulogu jer povećavaju preciznost, smanjuju vibracije, poboljšavaju kontrolu momenta i omogućuju fleksibilnost u aplikacijama.

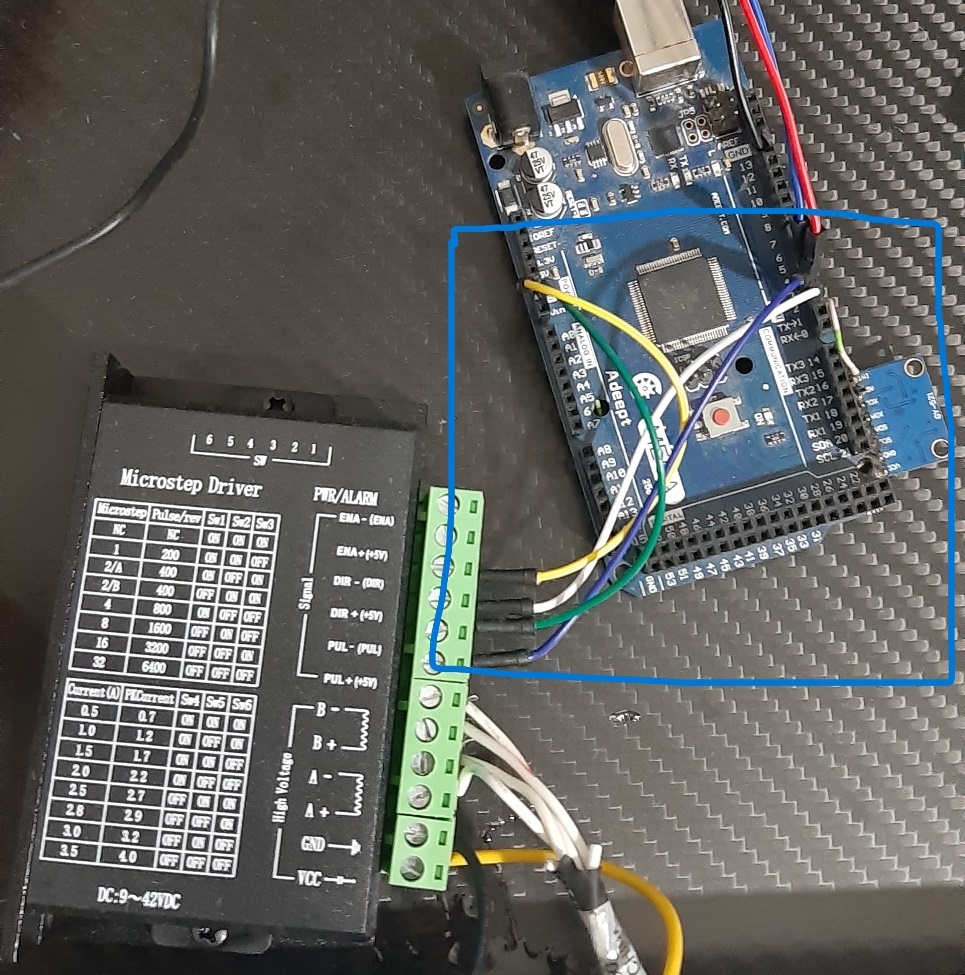
Na mikrostep drajveru nalaze se četiri pina koja se spajaju sa koračnim motorom. Koračni motor ima dva namotaja, A i B. Pinove **A+** i **A-** se spajaju sa prvim namotajem A, **B+** i **B-** sa namotajem B. Također na mikrostep drajver potrebno je spojiti sa izvorom od 12 V (žuta žica) i uzemljenjem (crna žica) od napojne jedinice



Slika 47. Spajanje koračnog motora i mikrostep drajvera

#### Spajanje mikrostep drajvera sa Arduinom

Mikrostep drajvere treba spojiti s Arduinom. Pinovi **DIR+** i **DIR–** koriste se za kontrolu smjera rotacije motora; pri čemu se DIR+ spaja na izvor od 5V, a DIR– na uzemljenje (ground). Drugi par pinova, **PUL+** i **PUL-**, služe za upravljanje brzinom rotacije motora. PUL+ se također spaja na 5V izvor, dok se PUL– povezuje na uzemljenje. Pinovi za omogućavanje (**ENA+** i **ENA-**) kontrolišu omogućavanje i onemogućavanje rada koračnog motora, ali pošto ta funkcija nije potrebna u ovom projektu, ti pinovi neće biti povezani s Arduinom. Pinove DIR+ i PUL+ treba povezati na odgovarajuće **general purpose** pinove na Arduinu.



Slika 48. Spajanje mikrostep drajvera i Arduina

### Kod za kretanje koračnih motora u Arduino razvojnom okruženju

Nakon što su izvršni dijelovi uspješno spojeni, potrebno je testirati rad konačnih motora i odrediti komande za kretanje. Da bi se to postiglo potrebno je napisati kod u Ardunio razvojnom okruženju koji će serijskom komunikacijom primati poruke, i na osnovu njih pokretati koračne motore u zadanim smjerovima. Serijske poruke će se slati sa Python programa.

#### Uspostavljanje komunikacije između koračnih motora

U Arduino razvojnom okruženju potrebno je uspostaviti serijsku komunikaciju. Zatim se definišu step i direction pinovi. Za lijevi koarčni motor na Arduinu definisan je pin 2 za direction pin, a pin 3 za step pin. Za desni koračni motor, pin 4 je direction pin, a pin 5 step pin.

Za pokretanje koračnih motora koriste se metode iz **AccelStepper.h** biblioteka. Ta biblioteka omogućuje kontrolu ubrzanja i usporenja, multitasking i podršku za više motora (do 10). U kodu se prvo definišu dva koračna motora. Zatim se odredi njihova maksimalna brzina i ubrzanje.



Slika 49. Definisanje koračnih motora u Arduino razvojnom okruženju

#### Definisanje komadi za kretanje mobilnog robota

Jedan koračni motor ima samo sva smjera kretanja, naprijed i nazad. Ali kombinacijom dva koračna motora moguće je postići i ostale smjerove kretanja kao što je rotiranje udesno i ulijevo, kao skretanje desno i lijevo. Kada se samo lijevi točak kreće naprijed ili desni točak unazad, robot se rotira u smjeru kazaljke na satu. A kada se desni točak kreće naprijed ili lijevi unazad, onda se robot rotira suprotno od smjera kazaljke na satu. Pri kretanju naprijed ili nazad, oba točka se okreću u istim smjerovima.

Funkcijama **moveTo** i **stop** se kontroliše kretanje koračnog motora. Ako je vrijednost unutar moveTo metode pozitivna, motor se okreće naprijed, a ako je negativna onda unazad. Odlučeno je da se vrijednosti 1000000 i -1000000 koriste za kretanje motora naprijed i unazad. Za pravce kretanja definisane su slijedeće komande:

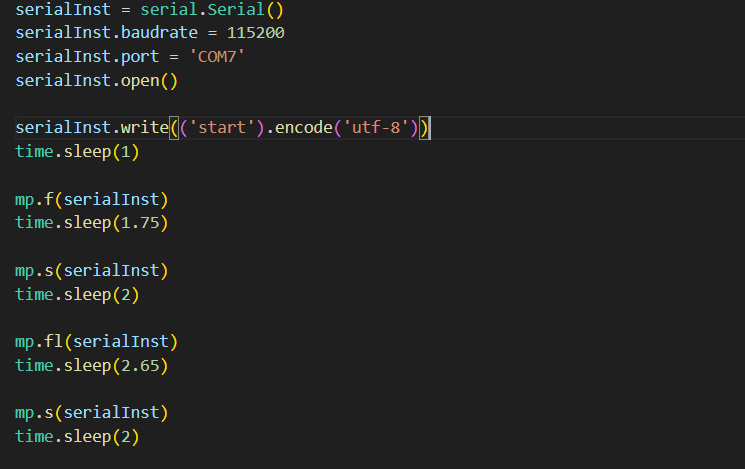
* **f** – kretanje naprijed (oba motora se okreću naprijed)
* **r** – kretanje unazad (oba motora se okreću unazad)
* **s** – zaustavljanje oba motora
* **f1** – kretanje lijevog motora naprijed
* **r1** – kretanje lijevog motora unazad
* **s1** – zaustavljanje lijevog motora
* **f2** – kretanje desnog motora naprijed
* **r2** – kretanje desnog motora unazad
* **s2** – zaustavljanje desnog motora



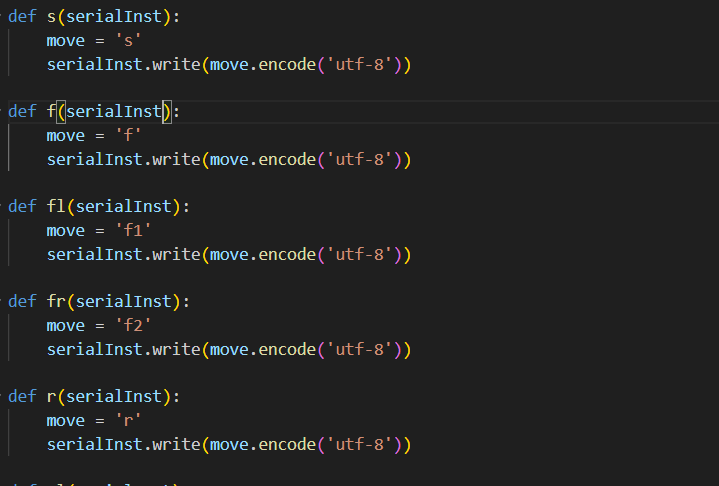
Slika 50. Definisanje komadi za kretanje koračnih motora

Nakon napisanog koda, kretanje točkova potrebno je i testirati. Ovaj dio je ključni za kretanje robota. Pored ispravnosti koda, bitno je i da su fizičke komponente pravilno povezane kako bi se točkovi okretali istom brzinom. Za testiranje, komade se šalju preko serijskog monitora u Ardunio razvojnom okruženje. Kada se komanda pošalje, pohranjuje se u varijablu i provjerava da li odgovara jednom od uslova. Testiranje je prošlo uspješno.

Nakon testiranja komadi kretanja sa serijskim monitorom, potrebno je ostvarit komunikaciju između Arduina i Python programa. U Python programu potrebno je ostvariti serijsku komunikaciju. Da bi se to postignulo, baud rate mora biti isti kao i u Arduino razvojnom okruženju, 115200. Nakon toga pozivaju se metode u kojim se šalju komande serijskom komunikacijom na Arduino. Metode su odvojene u zaseban Python fajl.



Slika 51. Ostvarivanje serijske komunikacije u Python programu



Slika 52. Metode za slanje komandi serijskom komunikacijom

### Određivanje dužine kretanja robota

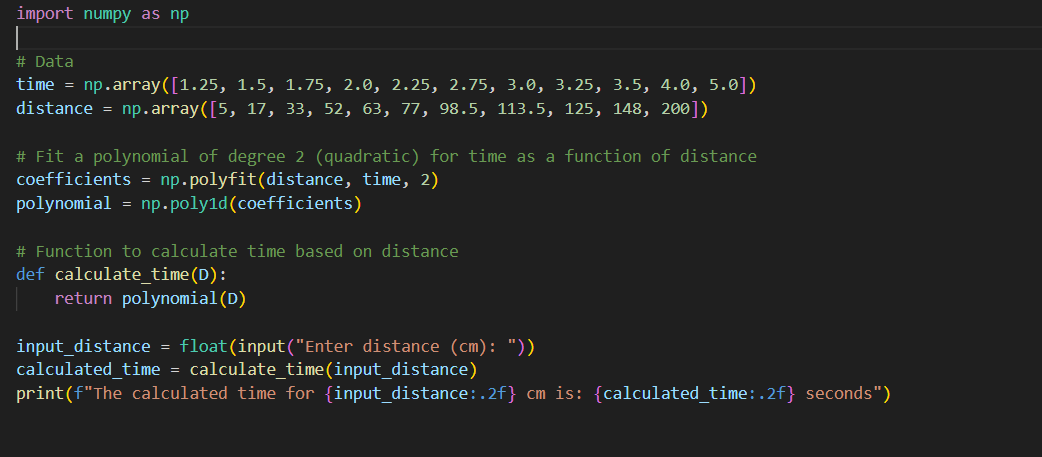
Nakon uspješne komunikacije Python programa i Arduina, potrebno je odrediti koliko vremena je potrebno da se određena komanda za kretanja izvršava da bi se prešla zadana distanca. Kontroliranje vremena izvršavanja komadni za kretanje u Pythonu vrši pomoću funkicje **sleep** iz modula **time**. Da bi se kreirala metoda koja na osnovu date distance vraća vrijem potrebni su podaci o mjerenjima.

#### Računanje vremena na osnovu distance za pravolinijska kretanja

Nakon uspješne komunikacije Python programa i Arduina, potrebno je odrediti koliko vremena je potrebno da se određena komanda za kretanja izvršava da bi se prešla zadana distanca. Kontroliranje vremena izvršavanja komadni za kretanje u Pythonu vrši pomoću funkicje **sleep** iz modula **time**. Da bi se kreirala metoda koja na osnovu date distance vraća vrijem potrebni su podaci o mjerenjima.

|  |  |
| --- | --- |
| Vrijeme (u sekundama) | Distanca (u cm) |
| 1.25 | 5 |
| 1.5 | 17 |
| 1.75 | 33 |
| 2.0 | 52 |
| 2.25 | 63 |
| 2.5 | 77 |
| 2.75 | 88 |
| 3.0 | 98.5 |
| 3.25 | 113.5 |
| 3.5 | 125 |
| 4 | 148 |
| 5 | 200 |

Na osnovu podataka prvo se izračunaju koeficijenti polinomijalne jednačine drugog stepena. Zatim se kreira polinomijalna funkcija, koja prima vrijednost distance, a vraća vrijeme u sekundama. Polinomijalna funkcija stavlja unutra metode koja će biti javno dostupna za korištenje u drugim Python fajlovima. Metoda je uspješno testirana. Za 77 cm izračunato je 2.58 sekundi (2.5 sekundi je u tabeli).



Slika 53. Kod za izračunavanje vremana kretanja za određenu distancu

#### Računanje vremena za rotaciju mobilog robota

Pored pravolinijskog kretanja kada se okreću oba koračna motora, postoje i rotaciona kretanja kada se okreće samo jedna koračni motor. Kod rotacije mobilnog robota, mjerna jedinca je stepen. U ovim radu koristi se samo rotacija za 90 stepeni, tako da će samo za taj slučaj biti određeno vrijeme. Pošto je stepen rotacije uvijek fiksan (90), nije potrebna tabela sa podacima kao kod pravolinijskog kretanja. Nakon testiranja, ustanovljeno je da se rotacija od 90 stepni postiže za vrijeme između 2.6 i 2.7 sekundi. Na osnovu toga odlučeno je da se za rotaciju od 90 koristiti vrijeme od 2.65 sekundi.

#### Upravljanje višestrukim komandama za kretanje

Izračunato je vrijeme za pravolinijska i rotaciona kretanja motra, ali samo u slučajevima kada se izvršava jedna komada. Potrebno je i provjeriti da li rezultati odbijeni iz pojedinačnog računanja važe i za višestruke komade, jer u većini slučajeva, robot će se kretati u više od jednog pravca, npr. kada se prvo potrebno kretati pravo, pa okrenutu za 90 stepeni udesno, i onda se opet kretati pravo.

Nakon testiranja i mjerenja, došlo se do zaključka da između izvršavanja višestrukih komadi postoji vremensko kašnjenje. Npr. u slučaju kada je za drugu komadu određeno da se izvršava 9 sekundi, zbog kašnjenja, komanda će se izvršavati 5 do 6 sekundi. Vrijeme kašnjenja bi također trebalo izračunati mjerenjem.

Pronađeno je jednostavnije rješenje. Umjesto mjerenja, nakon svake komande za kretanje će se pozvati komada za zaustavljanje jednog li dva motora, u ovisnosti da li je kretanje pravolinijsko ili rotaciono. Na taj način svaka naredna komanda za kretanje se ponaša kao da je ona prva. Tako sa za ovaj način vrijede prethodan računanja i potrebno mjeriti i računati kašnjenja.

## Integracija svih dijelova mobilnog robota

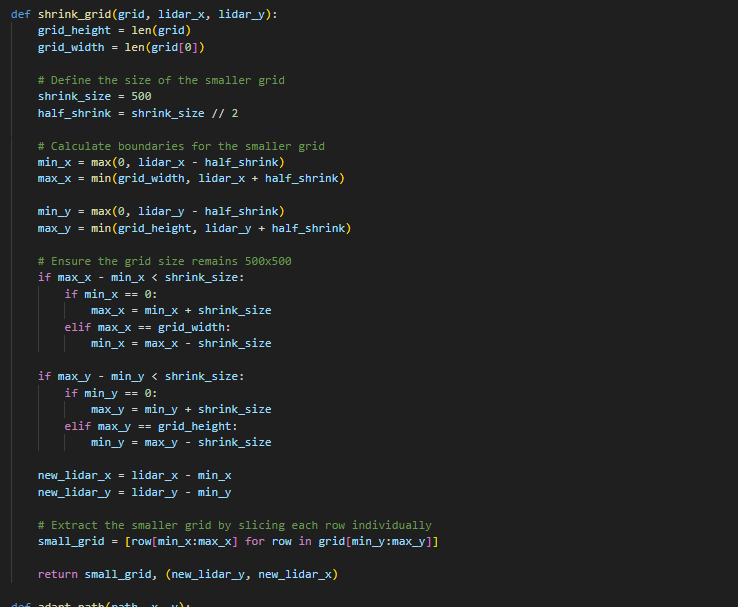
Nakon što su dijelovi mobilnog robota (Lidar, Python program i izvršni dio) završeni i uspješno testirani pojedinačno, potrebno je sve dijelove spojiti u jednu cjelinu. Iako dijelovi uspješno rade pojedinačno, to ne znači da će raditi i kada se spoje. Moguće su i potencijalne korekcije i dorade.

### Prilagođavanje prostora

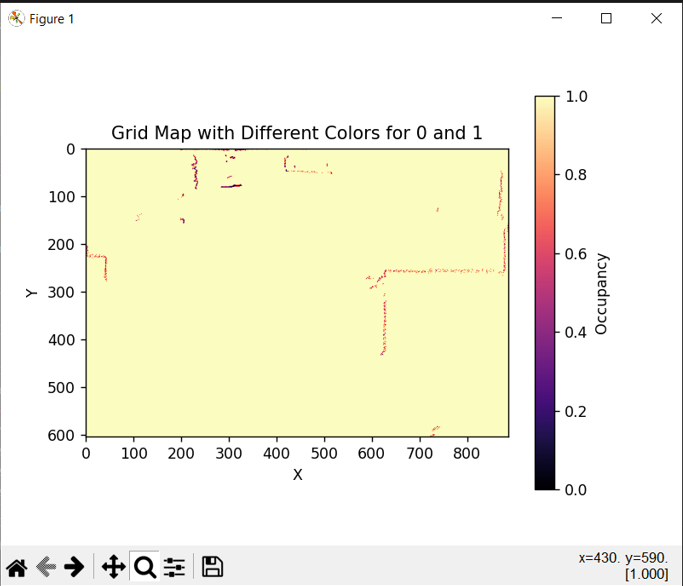
U nekim slučajevima, algoritam nije mogao izračunati najkraću putanju, i vraćana je greška. Nakon analize, ustanovljeno je da algoritam ne može izračunati najkraću putanju u prostoru koji ima veću širinu ili dužinu. Da bi se algoritam ipak mogao izvršavati u svim prostorima, bilo je potrebno smanjiti mapirani prostor. Odlučeno je da se uzme prostor koji je za 250 cm udaljen u svim pravcima od pozicije Lidara. **Površina jedne tačke u prostoru, odnosno jednog elementa u gridu mapiranog prostora u stvarnom okruženju iznosi 1 cm2**.

Za smanjivanje grida kreirana je metoda. Metoda kao ulazni parametar prima početni grid koji je potrebno smanjiti. Prilikom smanjivanja grida potrebno je provjeriti da od pozicije Lidara u svim pravcima postoji 250 mjesta, za svaki pravac pojedinačno. Npr. ako je širina prostora 450 cm, a x koordinata pozicije Lidara 300, u desnom pravcu moguće je uzeti samo 150 cm. Ako se pokuša uzeti 250 cm, desit će se greška jer će se izaći iz granica prostora.

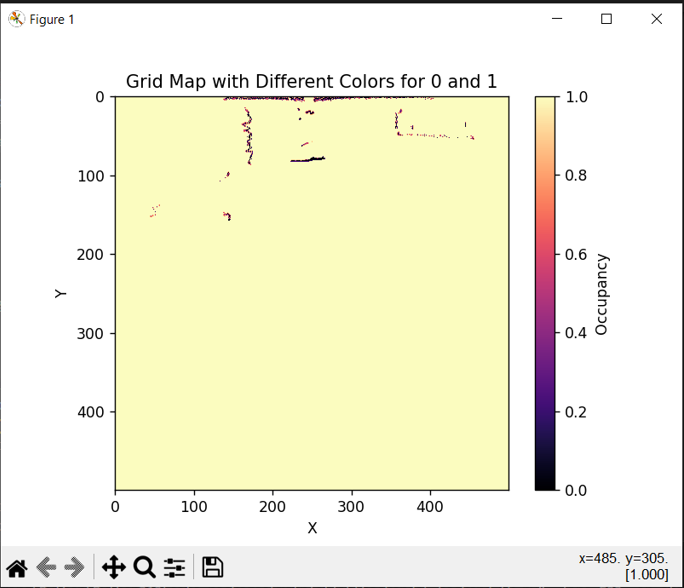
Ispod se može vidjeti slika koda metode za smanjivanje grida, i sike prvobitnog grida i smanjenog grida. Na slici prvobitnog grida dimenzije su 600 x 900, a na slici smanjenog grida 500 x 500, što pokazuje da metoda uspješno smanjuje grid.



Slika 54. Metoda za smanjivanje grida



Slika 55. Prvobitni grid

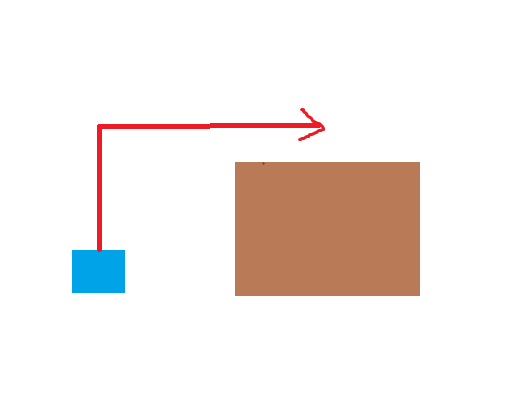


Slika 56. Smanjeni grid

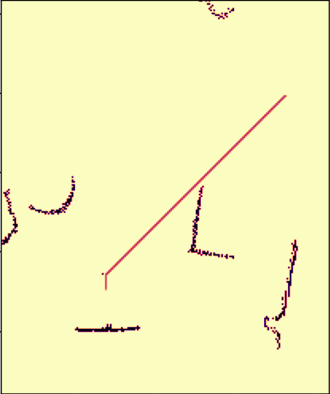
### Prilagođavanje putanje

Putanja koja je izračunata, obično sadrži veliki broj promjena smjera kretanja. To nije poželjno, jer robot zbog svojih dimenzija ne može često, nakon par centimetar pređenog puta, mijenjati smjer kretanja. Zbog toga je potrebo uraditi prilagođavanje putanje.

U ovom radu uradit će se prilagođavanje putanje u slučaju kada se robot nalazi paralelno s preprekom i kada se ciljna pozicija udaljena naprijed i desno od njegove početne pozicije. Primjer takve putanje je prikazan na slici ispod.

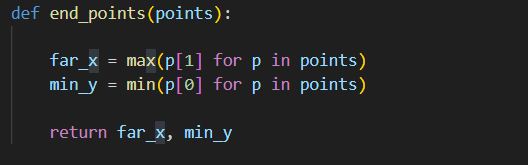


Slika 57. Skica putanje koja će se prilagođavati



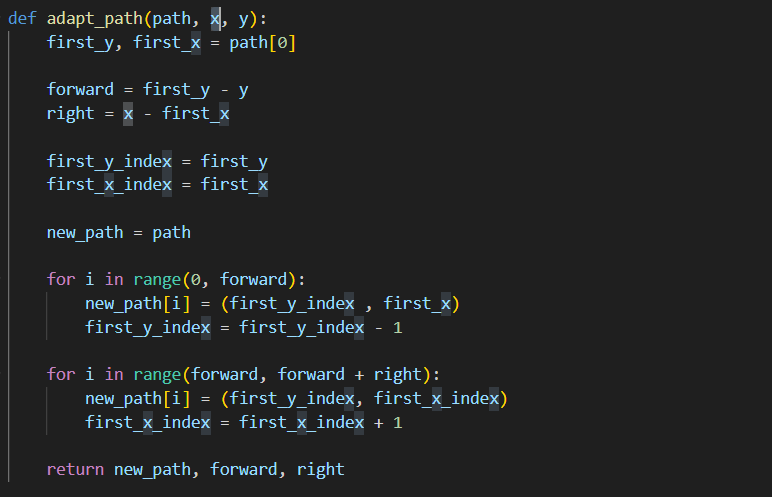
Slika 58. Slika prvobitne putanje

Na slici iznad je vizuelno prikazana prvobitna putanja koju je potrebno adaptirat. Zato što je ciljna pozicija udaljena naprijed i desno od početne pozicije, potrebno je pronaći najveću vrijednost x koordinate koja predstavlja najdalje pomicanje u desnom pravcu i najmanju vrijednost y koordinate koja prestavlja najdalje pomicanje naprijed (najmanju zato što se pri kretanje naprijed smanjuje vrijednost koordinate na y osi). Za pronalazak krajnjih tačaka kreira se zasebna metoda.



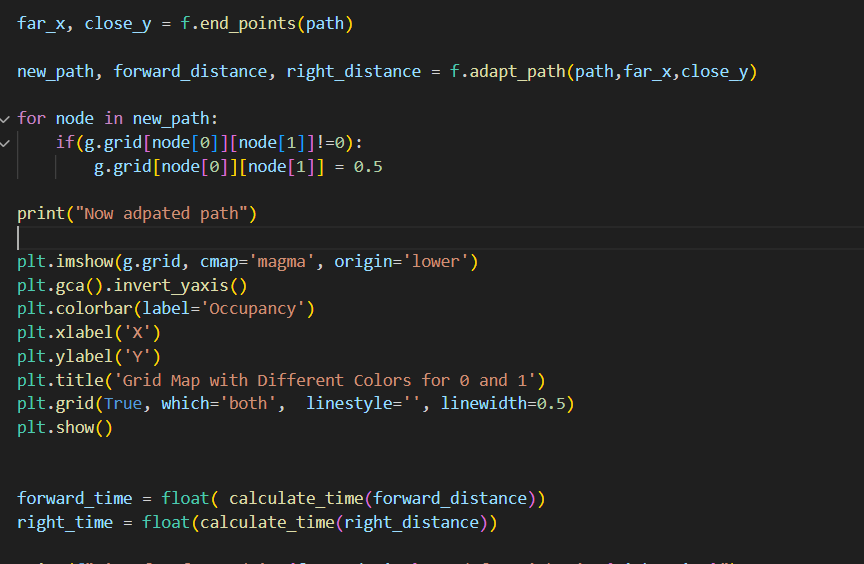
Slika 59. Metoda za pronalazak kranjih tačaka

Za prilagođavanje putanje se također kreira zasebna metoda koja kao ulazne parametre prima prvobitnu putanju i krajnje tačke naprijed i desno. Zatim se izračunava dužina kretanja naprijed tako što se od krajnje tačke naprijed oduzme y koordinata početne pozicije Lidara. Isti postupak je i za kretanje udesno. Kreira se nova putanja. Prvo se ide naprijed dok se ne dođe do krajnje tačke naprijed. U tom procesu x vrijednost je uvijek ista, jednaka x vrijednosti početne pozicije Lidara. Zatim se ide desno. U tom procesu y ostaje isti sa vrijednošću krajnje tačke naprijed.

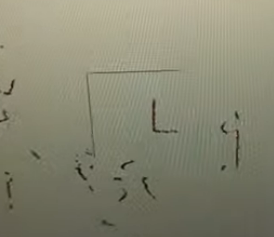


Slika 60. Metoda za prilagođavanje putanje

Nakon što se prvobitna putanja izračuna, poziva se funkcija za prilagođavanje putanje. Prilagođena putanja se prikazuje i računa se vrijeme potrebno za izvršavanje komande za kretanje naprijed i desno na osnovu prethodno izračunatih distanca.

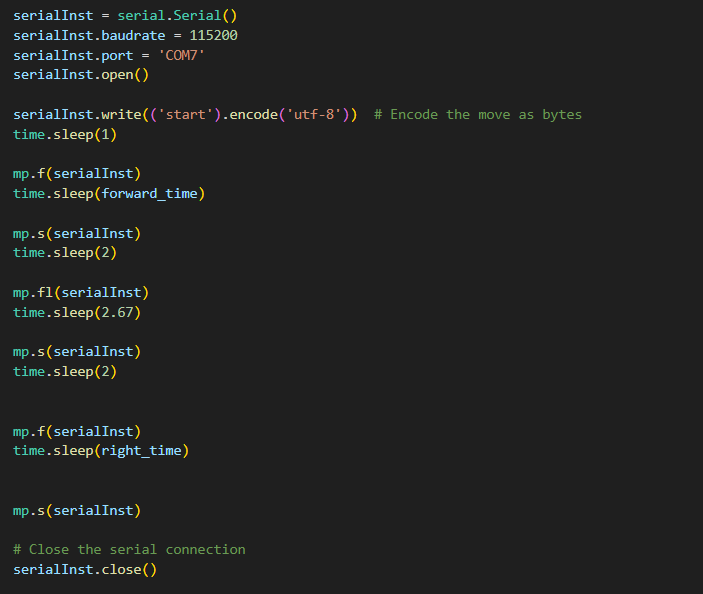


Slika 61. Implementacija metode za prilagođavanje putanje



Slika 62. Prikaz prilagođene putanje

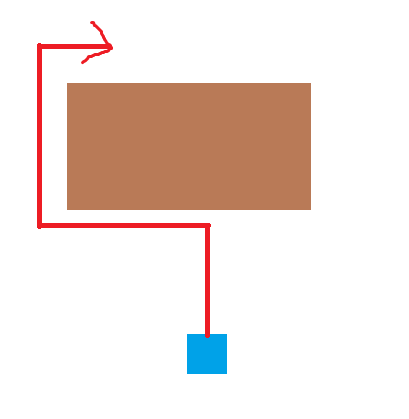
Nakon prikaza prilagođene putanje, otvara se serijska komunikacija. Prva komada za kretanje je **naprijed**, pa se poziva funkcija „**f“** sa vremenom izvršavanja koje je prethodno izračunato za kretanje naprijed. Zatim se skreće desno, pa se poziva komanda **„fl“** koja se izvršava 2.67 sekundi što je potrebno za rotaciju od 90 stepeni. Zadnja komanda kretanja je opet **naprijed**, sa vremenom izvšavanja koje je prethodno izračunato za kretanje udesno. Na kraju serijska komunikacije se zatvara, a robot je stigao do ciljne pozicije.



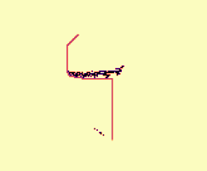
Slika 63. Kod za kretanje robota po prilagođenoj putanji

### Prilagođavanje kompleksne putanje

Prethodno je prilagođena jednostavnija putanja. Pri kretanju mobilnog robota moguće su i kompleksnije putanje. Primjer takve putanje se nalazi na slici ispod, kada se prepreka nalazi ispred robota, a ciljna pozicija se nalazi iza prepreke.

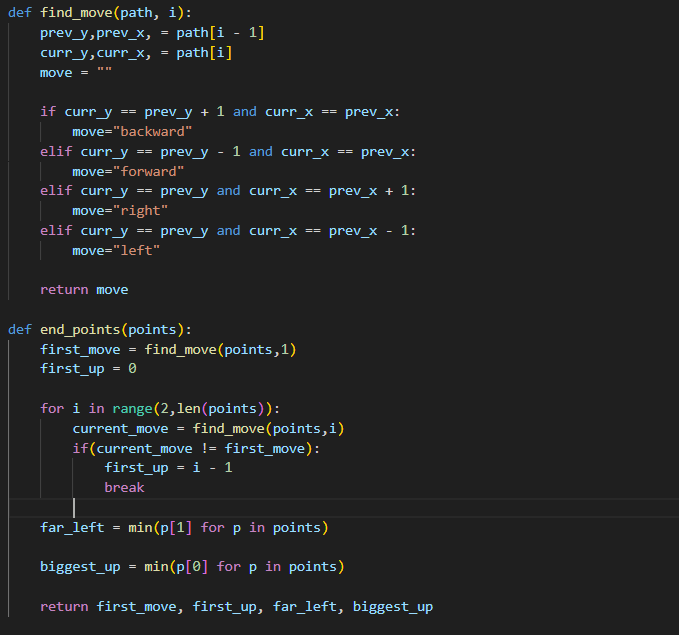


Slika 64. Skica kompleksnije putanje



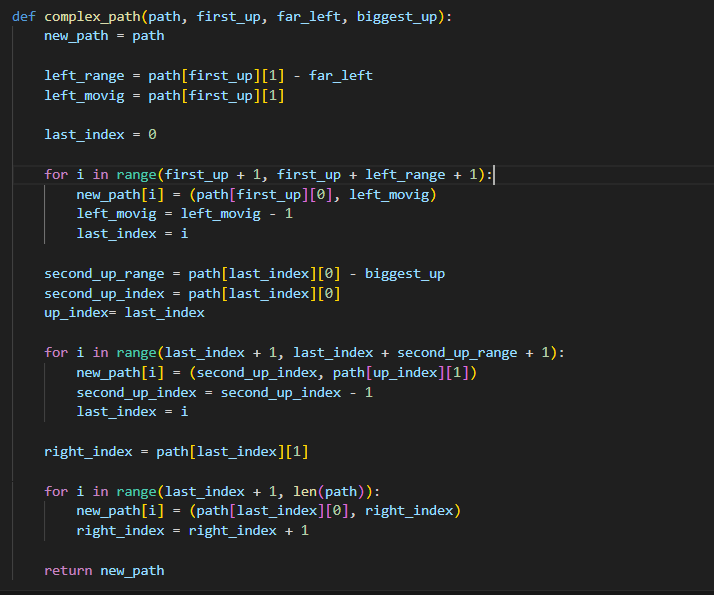
Slika 65. Prvobitna kompleksna putanja

Kao i kod jednostavnije putanje, potrebno je smanjiti broj promjena smjera kretanja, što će se postići kreiranjem metode. Ali prije toga potrebno je pronaći krajnje tačke. Putanja se treba prilagoditi da se robot kreće slijedećim redoslijedom: naprijed, lijevo, naprijed i desno. Potrebno je pronaći krajnje tačke za sva ta kretanja, što će se postići pomoćnim metodama na slici ispod.



Slika 66. Pomoćne metode za prilagođavanje kompleksne putanje

U metodu se proslijede prvobitna putanja i krajnje tačke. Kreira se niz koji će sadržavati tačke prilagođene putanje. Prvo kretanje naprijed je isto kao i kod prvobitne putanje. Od te pozicije se kreće lijevo sve do krajnje tačke lijevo. Nakon toga se ponovo kreće pravo i ide do krajnje tačke za drugo kretanje pravo. I na kraju se ide desno do krajnje tačke desno. Nakon testiranja ova metoda se pokazala uspješnom.



Slika 67. Metoda za prilagođavanje kompleksne putanje



Slika 68. Prikaz uspješno prilagođene kompleksne putanje

## Zaključak

U ovom diplomskom radu detaljno je opisan razvoj mobilnog robota koji koristi Lidar senzor i A\* algoritam pretrage za izbjegavanje prepreka. Ovaj rad predstavlja značajan doprinos u oblasti autonomnih sistema. Implementacija Lidara omogućava robusno i precizno mapiranje prostora, pružajući robotu podatke o udaljenosti i prepreka u realnom vremenu. A\* algoritam, s druge strane, koristi ove podatke kako bi izračunao najkraću putanju prema cilju, uzimajući u obzir kako trenutne prepreke tako i heurističku procjenu udaljenosti do ciljne tačke.

Predstavljeni sistem pokazuje sposobnost pronalaska putanje kroz kompleksne prostore, prilagođavajući se dinamičnim uslovima. Zahvaljujući kombinaciji algoritma umjetne inteligencije i preciznih senzorskih podataka, ovaj robot može učinkovito prepoznavati prepreke i mijenjati svoju putanju kako bi se izbjegli sudari. Razvijen sistem ne samo da nudi visok nivo autonomije, već se može primijeniti u različitim scenarijima, od unutrašnje robotike do vanjskih, zahtjevnijih okruženja.

Ovaj rad dokazuje da upotreba senzora i algoritma poput A\* algoritma pretrage može značajno unaprijediti performanse robota u stvarnim situacijama, povećavajući efikasnost navigacije i smanjujući vrijeme potrebno za postizanje ciljeva.

### Buduća unapređenja

Ovaj mobilni robot je moguće unaprijediti na slijedeće načine:

* **Integracijom dodatnih senzora**: Korištenje dodatnih senzora, poput vizualnih kamera ili infracrvenih senzora, može omogućiti robusnije prepoznavanje prepreka i objekata u okruženju.
* **Optimizacijom algoritama**: Iako je A\* algoritam efikasan, njegovu brzinu i performanse moguće je dodatno unaprijediti primjenom naprednijih algoritama poput D\* Lite, koji je prilagođeniji za dinamička okruženja.
* **Unapređenjem prilagodljivosti sistema**: Povećanje fleksibilnosti sistema omogućavanjem robota da se automatski prilagođava različitim vrstama terena ili preprekama dodatno bi povećalo njegove sposobnosti.
* **Energetskom efikasnošću**: Dalja optimizacija upotrebe energije može doprinijeti dužem trajanju baterije i većoj autonomiji robota, čineći ga korisnijim za duga operativna razdoblja.

## Literatura

1. Koračni motori <https://cnc-centar.ba/aktuelnosti/sta-je-step-koracni-motor/>
2. Napojna jedinica <https://hardwercom.wordpress.com/2018/03/23/napojna-jedinica/>
3. Mikrostep drajver <https://www.electromate.com/mechatronic-automation/mechatronic-automation-components/stepper-drives/microstep-drivers/>
4. Arduino <https://starmotech.com/sta-je-to-arduino/>
5. Dokumentacija o Lidaru <https://github.com/ldrobotSensorTeam/DeveloperKit/releases>
6. GitHub profil Lidara <https://github.com/ldrobotSensorTeam>
7. Guthub profil na kojem se nalzai kod u Pyhonu koji dohvata tačke prepreka <https://github.com/ldrobotsensor>
8. A\* algoritam pretrage i njegov prvi kod koji je testiran https://www.geeksforgeeks.org/a-search-algorithm/
9. Repostiory u kojem se nalazi kod A\* algoritma pretrage koji je korišten u radu <https://github.com/mlyean/grid-pathfinding>