SZAKDOLGOZAT

Lehoczki Viktor

Gépipari automatizálási szakmérnök

Gödöllő

2023

Magyar Agrár- és Élettudományi Egyetem
Szent István Campus
Műszaki Intézet
Gépipari Automatizálási Szakmérnöki Szak

Mozgórobot navigációs algoritmusának fejlesztése ROS2 környezetben

Belső konzulens:	Tóth János
	egyetemi tanársegéd
Külső konzulens:	Szabó Bence Zsolt
	Robot programozó mérnök
Készítette:	Lehoczki Viktor
	DEJZQJ
	Levelező tagozat
Intézet/Tanszék:	Műszaki Intézet
	Mechatronikai Tanszék

Gödöllő

2023

MAGYAR AGRÁR- ÉS ÉLETTUDOMÁNYI EGYETEM

Szent István Campus, Gödöllő
Cím: 2100 Gödöllő, Páter Károly utca 1.
Tel.: +36-28/522-000
Honlap: https://godollo.uni-mate.hu

MŰSZAKI INTÉZET
GÉPIPARI AUTOMATIZÁLÁSI SZAKMÉRNÖK

DIPLOMADOLGOZAT

feladatlap

Lehoczki Viktor *(DEJZQJ)*

részére

A diplomadolgozat címe:

Mozgó robot navigációs algoritmusának fejlesztése ROS2 környezetben

Feladatkiírás:

Bevezetés, Szakirodalom feldolgozása, Probléma bemutatása, Mozgó robot navigációs algoritmusának fejlesztése, A pontból B pontba történő eljutása Gazebo szimulációs környezetben, Továbblépési lehetőségek, Gazdasági számítás, Összefoglalás

Közreműködő tanszék: Mechatronika

Külső konzulens: *Szabó Bence Zsolt, robot-programozó mérnök, Flame Spray Hungary Kft.*

Belső konzulens: *Tóth János, egyetemi tanársegéd,* MATE, Műszaki Intézet

Beadási határidő: 2023. november 06.

Gödöllő, 2023. szeptember 04.

Jóváhagyom		**Átvettem**
(tanszékvezető)	(szakfelelős)	(hallgató)

A dolgozat készítőjének külső konzulense nyilatkozom arról, hogy a hallgató az előre egyeztetett konzultációkon megjelent.

Gödöllő, 2023. 11 hó 10 nap

(külső konzulens)

Tartalomjegyzék

Szent István Campus, Gödöllő
Cím: 2100 Gödöllő, Páter Károly utca 1.
Tel.: +36-28/522-000
Honlap: https://godollo.uni-mate.hu

1. Bevezetés

Napjainkban, a robottechnológiában alkalmazott navigációs rendszerek elengedhetetlenek az autonóm robotok kialakításához. Az autonóm robotok navigációs rendszere három alapvető folyamatból áll: térképezés, lokalizáció és útvonaltervezés. A térképezés során a robot egy távolságmérő eszköz, például egy lézerszkenner segítségével ismeri fel és rögzíti a környezetét. A lokalizáció a térképezés során gyűjtött adatokat használja fel a robot jelenlegi helyzetének meghatározására a térben. Az útvonaltervezés pedig a robot aktuális helyzetét és a környezeti adatokat használja fel az ütközésmentes útvonal kialakításához. A navigációs rendszerek széles körben alkalmazzák, például önvezető autókban, mobil ipari robotokban és robotporszívókban is.

Szakdolgozatom témája a mozgórobot navigációs algoritmusának fejlesztése ROS2 környezetben. Dolgozatom célja, a mozgórobot sík felületen történő mozgása a kijelölt kezdőpontól, egészen a végpontig, különböző akadályokat elkerülve a megtett pálya során. A dolgozatomban egy járőr mozgórobot alkalmazását készítem el. Először az autonóm navigációt hoztam létre a mozgórobot számára majd az általam C++ megírt indítóprogramok segítségével a meghatározott 4 pontban a robot folyamatos önjárást végez.

A robotok önálló mozgását a navigációs rendszer megvalósításával érhetjük el. A navigációs rendszer folyamatai egymással összefüggenek. A térképezési folyamatot a lokalizációs folyamat előtt kell végrehajtani. Az útvonaltervezési folyamat nem hajtható végre térképezési és lokalizációs folyamat nélkül. A térképezési és lokalizációs folyamatokat egyidejűleg is el lehet végezni a Simultaneous Localization and Mapping (SLAM) segítségével. A navigációs rendszer befejezéséhez az útvonaltervező rendszert kell végrehajtani. A szakdolgozatom elkészítéséhez Linux operációs rendszert használtam, így stabil futtást eredményezett a Gazebo szimulációs szoftvernek és a ROS2 keretrendszernek egyaránt. A Gazebo támogatja a ROS keretrendszert, ami a legstabilabb verzió, mivel teljes mértékben hozzájárul az Ubuntu LTS-jéhez (Long Term Support).

Szent István Campus, Gödöllő
Cím: 2100 Gödöllő, Páter Károly utca 1.
Tel.: +36-28/522-000
Honlap: https://godollo.uni-mate.hu

2. Szakirodalom áttekintés

Ebben a szakaszban bemutatom a mobil robotok navigációs problémáinak korábbi megoldásait valamint az ezen a területen végzett előző kutatásokat. Az elmúlt évtizedek során megannyi kutatás foglalkozott ezzel a témával. Habár számos technikát dolgoztak ki a probléma megoldására, egyik sem vált általánosan elfogadottá. Ebben a részben áttekintem a mobil robotok navigációjában alkalmazott legfontosabb kutatási technikákat, eredményeket.

2.1. A mozgási mechanizmusok és kerék típusok

Az egyik helyről a másikba eljusson, egy mozgórobot, ahhoz konkrét mozgási szerkezetre van igénye. Ebből kifolyólag mozgási mechanizmus megválasztása fontos szempont a mobil robotok tervezése folyamán, figyelembe véve a sokszínű lehetséges helyváltoztatási technikára. A leggyakrabb ilyen helyváltoztatási mechanizmus, mint például a csúszás mint kígyó mozgás, a gyaloglás ember átall, ugrás a kenguru mozgása esetén, vagy az ugrás a kenguru mozgása által, ezek a mobil robotok számára is alkalmazható képesség. Ennek ellenére a biológiai architektúrák strukturális reagálásának létrehozása az ember révén rendkívül nehéz valamennyi ok miatt, többek között [41], [49], [52]:

- Mechanikai bonyolultság (mindegyik alkotóelem külön kell előállítani)
- Biológiai energiatároló rendszerek, melyek állatok által használt
- Matematikai komplexitás (ember alkotta rendszer mozgáselemzése akkor bonyolult, ha egy biológiai rendszer számos részből áll)

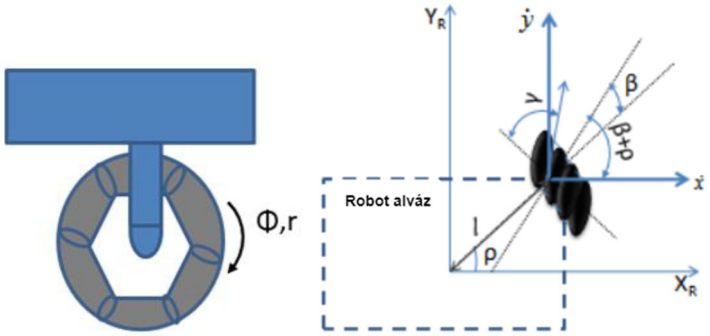
E szempontok miatt, amelyet széles körben is használnak, azok melyek kis számú ízülettel rendelkező biológiai mozgási rendszerből kell létrehozni. A lábon való mozgás nagy szabadságfokot igényel, ezáltal fő mechanikai komplexitás. Ennek kiküszöbölésének érdekében, aktív hajtású kerekekkel szerelik fel a mozgó robotokat. Ugyanakkor a mechanizmusnak az egyedüli korlátja, amikor a környezet (felület) nagyon „puha”, akkor nem társul hatékony eredmény, a gördülési súrlódás miatt (a kerék érintkezési pontja és a felület között). A mobil robotok irányítására számos ismert algoritmust és technikát ajánlottak, [41], [49], [52]. Öt típusba lehet sorolni a mobil robot kerekeit geometriai korlátjaik alapján [15].

Az egyszerű szerkezetük és megbízhatóságuk révén széles körben használják a kormányzott standard kerekkel ellátott mobil robotokat. Cariou részletezte a kormányzott kerekes robot kerekeinek kerékcsúszását és a kinematikai összefüggéseit az azonnali forgási központok helyzetének. Négykerekű kormányzott jármű számára készítettek egy automatikus

Szent István Campus, Gödöllő
Cím: 2100 Gödöllő, Páter Károly utca 1.
Tel.: +36-28/522-000
Honlap: https://godollo.uni-mate.hu

útvonalat, ahol mind oldalirányú, mind szögeltérésű csúszást vettek figyelembe. A görögős kerekeket használják túlnyomó többségben a háztartási termékekben például az irodai és otthoni használatokra, mivel a kormányzott kerekes mobil robotnak mobilitási korlátai vannak [23].

A svéd kerék felépítését nézve egy általános keréknek minősül, melynek a peremein görgők helyezkednek el. A robot teljes mobilitást biztosít a síkfelületen, a görgők pozíciónak köszönhetően, tehát ez eredményezteti, hogy akármilyen irányban képez mozgást végezni. Így az előbbi megállapításból elmondható, hogy a svéd kerekekkel felszerelt robot, mindenirányú mobil robotnak minősíthető. A svéd kereket az 1. ábra szemlélteti.

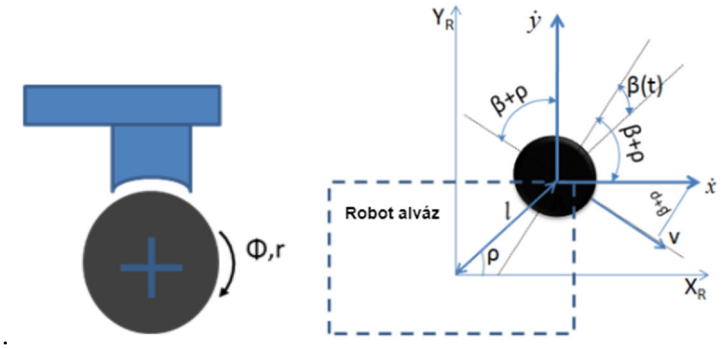


1.ábra: Svéd kerék és paraméterei [15]

Az N számú svéd kerekű mobil robot kinematikai modelljét illetve a mozgásvezérlési problémát Indiveri mutatta be. A svéd kerekkel építtetett mobil robot részére , egy útkövető szabályozási törvényt alkottak meg. Doroftei megalkotta az Omni robotot, egy mecanum kerekes mobil robotot, amely négy svéd kerékkel rendelkezett, ezáltal hagyományos kormányrendszer nélkül lehetet irányítani. [16],

A gömb alakú kerék, a kerekeknek egy másik csoportja. Ezek a kerek nagyobb stabilitást és mobiltást biztosítanak egy mobil robot számára, azonban mozgástervezésük illetve vezérlésük kellőképen összetett. A mindenirányú mobil robotokhoz, egy új kerék fejlesztését prezentálták, így képes volt felmászni a lépcsőn a mobil robot a félgömb alakú kerekeinek köszönhetően, a gömb kerekék felépítését a 2. ábra mutatja be. [15]

Szent István Campus, Gödöllő
Cím: 2100 Gödöllő, Páter Károly utca 1.
Tel.: +36-28/522-000
Honlap: https://godollo.uni-mate.hu

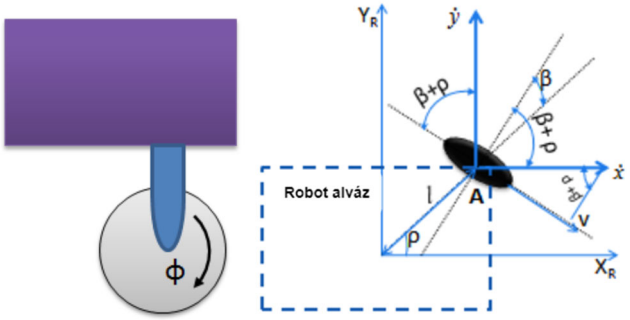


2. ábra: Gömb kerék és paraméterei [15]

A gömb és az Omni kerék közös felépítést is szemléltették, működési elvét tekintve két egymásra merőleges omni kerékpár hajt meg egy gömb alakú kereket.

„Gömblemez-problémának" hívják a gördülő gömb mozgástervezési problémáját és két eltérő algoritmust ajánlottak az újra konfiguráláshoz. Az első algoritmus Gauss-Bonet tételen alapul, amely azt mondja ki, hogy gömbháromszög manőverekkel lehet megvalósítani az újra konfigurálást. A második algoritmus váltakozó bementeket hív meg és általános kinematikai modellen alpszik, amely egyenes szakaszokból és körívekből álló megoldást hozzon létre. Kvatrerniók felhasználásával dolgozták ki a robot orientációjának leírását, a gömb alakú, kerekes robot kinematikai modellek számára. Egy egyetlen gömb kerekű mobil robot általános szerkezetét Lauwers mutatta be, ami bármilyen irányban képes volt mozogni [46].

A következő nagy kerék család típus, a standard kerekek, amelyeknek nincs függőleges forgástengelyük a kormányzáshoz. A rögzített szabványos kereket 3. ábra szemlélteti. Mivel az alvázhoz mért szöge rögzített, így az alaplappal való érintkezési pontja korlátozódik. Merthogy egy kerekes mobil robotnak három nélkülözhetetlen tulajdonsággal kell rendelkeznie, az első a manőverezhetőség, a második a stabilitás és végül az irányíthatóság, így legalább két kerék elegendő a statikus stabilitáshoz. Tömérdek kutatási munka irányult a kétkerekű differenciálműves robotra, mivel el tudd érni statikus stabilitást, ha a tömegközéppont a tengely alatt van. Hatab és Dhaouadi létre hozták a differenciálhajtású mobil robotdinamikát, mely segítséget nyújtott a kutatóknak a mobil robot a pályakövetéhez, a navigációhoz, valamint a megfelelő vezérlők tervezésében és modellezésében [15], [37].



3. ábra: Rögzített szabványos kerék és paraméterei [15]

Egy differenciálhajtású mobil robothoz, egy nemlineáris visszacsatolási útvezérlőt mutattak be, kinematikai modellt dolgoztak ki a differenciálrobot vezérlésre, majd ezt követően fuzzy logika segítségével valósították meg a szabályozási stratégiát. Menn bemutatott a csuklós több monociklusos mobil robotokhoz, egy általános kinematikai modellezési megközelítést. Majd ezután, gyakorlatba ültették ezt a módszertant a RobuRoc mobil robotra [15], [16]

2.2. Navigáció a mobil platform

A mozgó robot navigációjának fő központi eleme, hogy a mozgás során megjelenő helyi akadályokat elkerülje miközben a cél felé folyamatosan halad tovább. Mivel ez az egyik legfontosabb kritérium a mobil robot navigációja során, vagy az ütközésmentes haladás, így számos kutatás foglalkozott ezzel a témakörrel, az elmúlt évtizedekben. Habár több eljárást is kidolgoztak e probléma kiküszöbölésére, viszont egyik megoldás sem vált általánosan elfogadottá. Azonban vannak olyan kutatások, amelyek a helyi és globális navigációt egyesíti és ezek a megoldások mutatkoznak a leghatékonyabbnak. Robot navigációnak nevezik általában az akadályelkerülést, így a következő szempontokat foglalja magában:

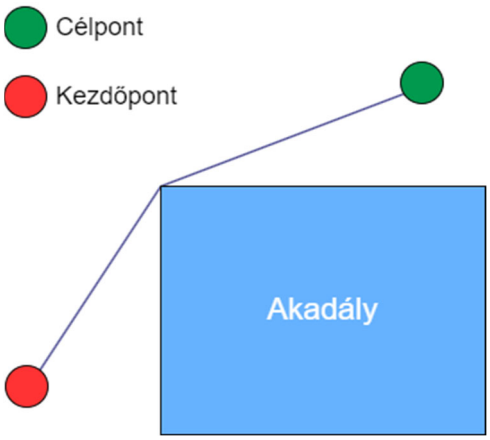
- Az megismerő térképezés „ a teljes környezet felismerése, melyet a helyek relatív helyzetének meghatározására valamint az útvonalak megtalálására használnak”
- A lokalizáció azt fogalmazza meg, hogy „ egy adott környezetben a robot hol helyezkedik el, az érzékelt információk szerint”
- Az útvonal tervezés azt jelenti „ a célok közötti szükséges kapcsolatot biztosít egy kijelölt cél elérésének érdekében”
- A mozgásvezérlés magába foglalja „ a robot mozgásának irányításával foglalkozik, annak eredményeképpen, hogy a mozgó robot elérje célját”

Ez a szakasz bemutatta a mobil robotok úttervezési aspektusait. Az útvonaltervezést két fő részre lehet osztani, helyi és globális úttervezésre. A robot előzetes ismeretekkel rendelkezik, a környezetében lévő akadályok alakjáról, elhelyezkedésről és még a mozgásról is. Számos stratégia alkalmazható a környezet felderítésre, különböző információkon keresztül, például a lézerérzékelők távolságadatai vagy a kamerák vizuális információi alapján. Elmondható, hogy a legtöbb ilyen megközelítés a mobil robotok navigációjának fő problémájával vizsgálódik [23].

2.2.1. Útvonaltervezés

Az autonóm robotok egy alapvető kritériuma az ütközésmentes útvonalak kialakításának képessége így az útvonaltervezés a robotika egyik kihívása. Az útvonaltervezés célja az útvonalak a megtervezése a környezet különböző pontjai között. Az útvonaltervezés hozzájárul a mobil robotoknak ahhoz, hogy mozgásuk során észleljék az akadályokat és megfelelő utat generáljanak az akadályok elkerülése érdekében [5].

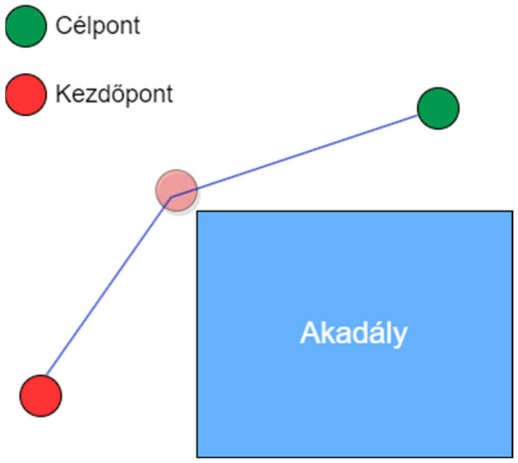
Az útvonaltervezés lényegi problémája az A pontból B pontba bevezető út megalkotása sík felületen egy adott környezetbe. Az A mobil robot jelenlegi pozíciója, a B. pedig a célpont. Pozíció mellett robotnak van orientációja is, tehát a probléma az, hogy a robot nem csak egy adott pont a térben. A B ponthoz való eljutáshoz a robotnak meg kell határoznia megfelelő irányt [6]. Robot geometriai adataira is szükségünk van az útvonaltervezés megalkotásához, így a robot sugarát be kell állítani az útvonaltervező számára. Máskülönben olyan utat hoz létre az útvonaltervező amely túl közel van az akadályhoz így a robotnak nincs elegendő helye a mozgáshoz tehát neki ütközik. A robot geometriája nélküli útvonaltervezést az 4. ábra mutatja, tehát a robot érinti az akadályt [8].



4. ábra: Útvonaltervezés robotgeometria nélkül [8]

Szent István Campus, Gödöllő
Cím: 2100 Gödöllő, Páter Károly utca 1.
Tel.: +36-28/522-000
Honlap: https://godollo.uni-mate.hu

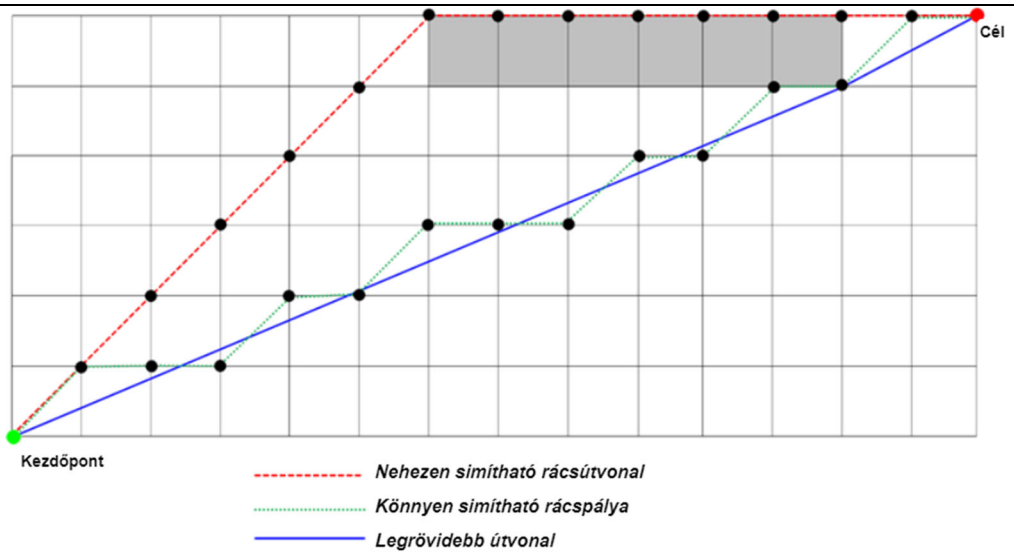
A robot sugarát bele kell számítani az útvonaltervezésébe, mivel a mobil robot középpontja az útvonalat jelenti. A robot geometriájának figyelembevételével történő útvonaltervezést a 5. ábra mutatja [8].



5. ábra: Útvonaltervezés robotgeometriával [8]

A gráf egy numerikus elképzelés, amelyet formálisan a csúcsok (vagy csomópontok) V elrendezése és a csúcsokhoz tartozó élek E elrendezése jellemez. Az éleket koordináltnak nevezzük, ha egy pár csúcs esetében egy él felhasználható az egyik csomópontból a következő csomópontba való eljutásra. Az éleknek lehet egy numerikus értékük is, amelyet gyakran súlynak vagy költségnek neveznek és amely az adott él mentén történő mozgáshoz szükséges "munka" elméleti ábrázolása [48].

Általában a gráf ábrázolása az egyik olyan módszer, amelyet a kutatók útvonaltervezésre használnak. A teret a rácshálóval mulattatják be, a rács méretét a kutató határozza meg. A rács mérete hatással van számítási időre és az útvonaltervezés eredményére is. Egy nagyméretű rácsnál a számítási idő megnő, mivel az útvonaltervezőnek az egész rácsot ki kell építenie, viszont részletes pályát eredményezhet mint a kisméretű rács estében. A gráf ábrázolására a 6. ábrán látható példa [21].



6. ábra: Az útvonaltervezés grafikonos ábrázolása [21]

2.2.2. Útvonaltervezési megközelítések

Az útiterv-módszerek alapvetően annak a környezetnek a dimenziójának csökkentésére szolgálnak, amelyben egy mobil robot navigációja és útvonaltervezése zajlik. Különböző útvonaltervezési algoritmusokat mutattak be a mobil robotok számára. Az algoritmusok eltérései az érzékelő típusa, a környezet, és a robot képességei szerint alakulnak. Az algoritmusok egy jobb és kifinomultabb teljesítmény nyújtanak, a költségek, az idő és a komplexitás tekintetében. Az útvonaltervezés általános meghatározása a legrövidebb és ütközésmentes út megtalálása. Több új kutatási módszer is jobb teljesítményt nyújt rövidebb útvonal meghatározásához, de nem garantálja hogy a rövidebb útvonal megalkotásához szükséges idő is csökkenne. Ezekben a kutatási módszerekben két jól ismert eljárás említhető meg a Voronoi-diagramok és a láthatósági grafikonok [31].

Voronoi diagramok

A voronoi diagram információkat rögzít a ponthalmazok közötti távolságokról bármely dimenziós térben. Úttervezéshez a Voronoi-t általában kétdimenziós térben alkalmazzák, ahol a pontkészletek egy síkon belül vannak. Egy síkot felosztanak cellákra úgy, hogy minden cella pontosan egy helyet tartalmazzon. A cella minden pontja esetén a pont euklideszi távolsága a cellán belüli helytől kisebb kell, hogy legyen, mint a pont távolsága a síkban lévő bármely más helytől. Ha ezt a szabályt a teljes síkon követjük, akkor a Voronoi-élként ismert cellák határai a pont egyenlő távolságát jelentik a

Szent István Campus, Gödöllő
Cím: 2100 Gödöllő, Páter Károly utca 1.
Tel.: +36-28/522-000
Honlap: https://godollo.uni-mate.hu

legközelebbi két helytől. Az általánosított Voronoi-diagram megtalálásához pontosan ki kell számítani a diagramot, vagy olyan közelítést kell használni, amely a Voronoi-diagram kiszámításának egyszerűbb problémáján alapul diszkrét pontok halmazára. O'Dunlaing és Yap általánosított voronoi diagram használatát javasolták az autonóm robotok mozgástervezéséhez. Később számos rögtönzött modellt javasoltak. Ilhan és Howie kidolgoztak egy új szenzoros útiterv-modellt, az úgyhívott generalizált Voronoi-gráf növekményes konstrukciót, amely egy már létező növekményes konstrukciós eljárásba került [6], [27]. Vlassis egy olyan módszert javasoltak, amely Voronoi-gráfot generált a robot által dinamikusan felépítve a konfigurációs terét egy adaptív klaszterezési algoritmus alkalmazásával a robot szabad terére. Kidolgoztak egy Voronoi-diagramot a robot szabad teréről a valószínűségi klaszterezési terv alapján. A valószínűségi növekvő sejtszerkezetek algoritmusa tartalmazza. [31]

- inicializálást,
- adaptációt,
- frissítést,
- klaszterbeillesztést
- klasztertörlést

Későbbi háromszögelési módszert alkalmaztunk a kapott klaszterközpontokra a gráf felépítéséhez. Lisien és Morales bemutattak egy új térképet, amelyet nagy szabad térben és esetleg magasabb dimenziókban működő robotokhoz terveztek, hierarchikus atlasznak nevezve. Ennek a hierarchikus atlasznak két szintje van: a topológiai térkép a legmagasabb szinten van, amely a szabad teret altérképekké alakítja az alsó szinten. Az alsóbb szintű altérképek egyszerűen funkciók gyűjteménye. Más feladatokhoz, például navigációhoz, a globális lokációhoz és az akadály elkerüléséhez, az eredményül kapott térkép szintén hasznos. Lee és Howie egy új útitervet vezettek be, amely egy konvex test irányítására szolgál egy ismeretlen síkbeli munkaterület felfedezéséhez, amelyet konvex hierarchikus általánosított Voronoi-gráfnak (konvex-HGVG) neveznek. A konvex-HGVG két komponensből áll:

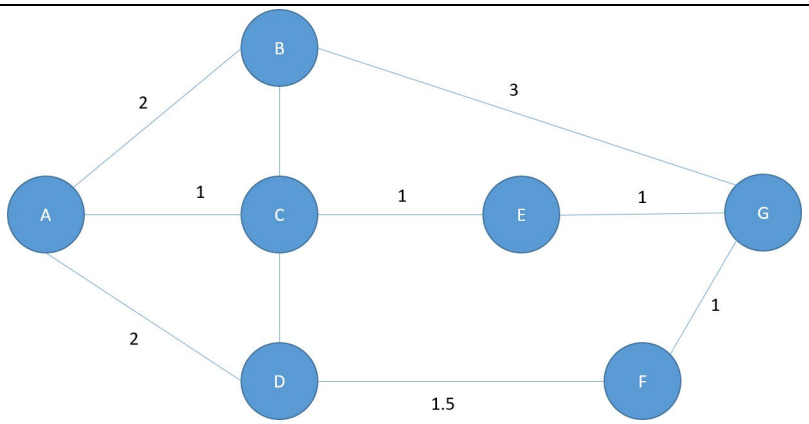
1. konvex-R élek, amelyek kétirányú, egyenlő távolságra lévő utak, amelyek szétkapcsolt konvex-GVG éleket kötnek össze
2. konvex-GVG élek, amelyek három irányban egyenlő távolságra vannak [24], [27].

Szent István Campus, Gödöllő
Cím: 2100 Gödöllő, Páter Károly utca 1.
Tel.: +36-28/522-000
Honlap: https://godollo.uni-mate.hu

Láthatósági grafikonok

A sokszög láthatósági gráfja egy olyan gráf, amely a sokszög élei az egymást látó csúcsok összekapcsolásával létrehozott sokszög éleinek felel meg és csomópontjai csúcsaihoz kapcsolódnak. A láthatósági grafikonok alkalmazásával megtalálhatjuk a legrövidebb euklideszi utakat a környezetben levő sokszögű akadályok között. Ezért az euklideszi legrövidebb út probléma a következőképpen oldható meg, a láthatósági gráf megalkotása, majd a legrövidebb út algoritmusa, például Dijkstra algoritmus alkalmazása a gráfra. Az akadályokhoz viszonyítva, nem elhanyagolható méretű robot mozgásának megtervezéséhez hasonló megközelítés használható, az akadályok kiterjesztése után, hogy beszámítsa a robot méretét. Az egyetlen akadály, egy átalános sokszög láthatósági grafikonjának pontjai, a sokszög csúcsai és a sokszög külső része. Hamilton gráfok az egyszerű sokszögek láthatósági gráfjai: a sokszög határa Hamilton-ciklust alkot a láthatósági gráfban, azonban ezeknek a grafikonoknak a pontos jellemzése nem ismert. Nyitott probléma ezen a területen még, hogy létezik-e olyan polinomiális időalgoritmus, amely képes-e a bemenetet gráfként felvenni és a láthatósági gráf poligonjaként állít elő kimenetet, feltételezve ha létezik ilyen sokszög.

Számos algoritmust dolgoztak ki a mobil robot útvonaltervezésére az egyik ilyen például az A*algoritmus és a genetikai algoritmussal történő útvonaltervezés. Az egyik olyan algoritmus a Dijkstra algoritmus [11] [35] [44]. Az algoritmus a csúcsok forrásától kezdve minden egyes szomszédos csúcs közötti távolság költségértékét hozzárendeli a csúcsok közötti távolsághoz. Amíg az összes csúcsot fel nem dolgozza az algoritmus, ez a lépés addig ismétlődik. A Dijkstra algoritmus végén a megoldás nem csak egyetlen legrövidebb út, hanem a forrás-csúcs és bármely más csúcs közötti távolság is. A Dijkstra algoritmusa kiszámítja a csúcsok közötti távolságot, ezzel lehetővé teszi, hogy a robot bármilyen megoldást használjon, amelyet a Djikstra algoritmusa biztosít. Minthogy az útvonaltervezés egyik fő célja a legrövidebb út megtalálása, a Dijkstra algoritmusa automatikusan kiválasztja a kis távolsággal rendelkező megoldást. A Dijkstra algoritmus gráfja a 7. ábrán látható [39].



7. ábra: A Dijkstra-algoritmus grafikonja [39]

A Dijkstra algoritmusának jobb verziója az A* algoritmus. Ez az algoritmus az Equation 3-on alapul [9].

$$f(n) = g(n) + h(n) \qquad (1)$$

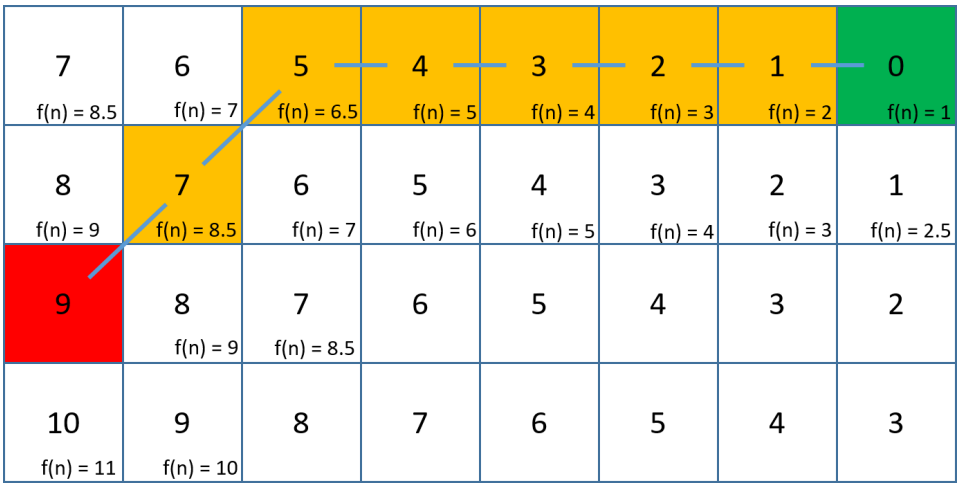
Ahol f(n) az n csomópont teljes költsége, g(n) az n csomópont elérésének költsége az aktuális csomópontról, h(n) a célcsomópont n csomópontról történő elérésének költsége. Az algoritmus kiszűri azt a csomópontot, amelynek legalacsonyabb f(n) értéke van. A lépést addig ismételjük, amíg az n csomópont nem lesz a célcsomópont [1]. Az A* algoritmus számítási ideje felülmúlja a Dijkstra algoritmusát.

Az algoritmus az összes csomópontot a memóriában tartja, ezáltal sok memóriát használ fel a folyamat elvégzéséhez. ez az A* algoritmus hátránya [43]. Sík területen a g(n) költségfüggvénye a robot mozgásától függ, például a függőleges és vízszintes mozgás költsége, az átlós mozgatás költsége 1,5 az algoritmus kiszámítja az aktuális csomópont körüli csomópontot. A számítási területet a 8. ábra szemlélteti [24].

Csomópont 1	Csomópont 2	Csomópont 3
Csomópont 4	Jelenlegi csomópont	Csomópont 5
Csomópont 6	Csomópont 7	Csomópont 8

*8. ábra: Az A*algoritmus számítási területe [24].*

Az algoritmus kiválasztja azt a csomópontot, amely a legalacsonyabb költséggel rendelkezik a többi csomópont közül, ezáltal ez a csomópont lesz a következő aktuális csomópont. A rendszer hozzáadja az előző csomópontot egy tömbhöz. A költségszámítást a 9. ábra mutatja [24].



9. ábra: Az A algoritmus költségkalkulációja [24]*

A csomópont és a célpozíció közötti távolságot a csomópont belüli szám adja meg. A piros csomópont a kiindulási helyzet, a zöld csomópont pedig a cél pont. A sárga csomópont az a csomópont, amely kis költséggel rendelkezik az előző csomóponthoz képest.Torres Haro véleménye szerint az A* algoritmus optimális útvonalat hoz létre, viszont időben költséges folymat. A g(n) és h(n) költségfüggvényét a környezettől függően ki kell igazítani, illetve dinamikus környezetben nem a legjobb választás A* algoritmus használata [13].

A genetikai algoritmus sokoldalú technikák gyűjteménye, amelyek felhasználhatók a fejlesztési és keresési kérdések kezelésére, beleértve a vizsgálandó tereket is. A keresést reprodukált fejlődés (a legalkalmasabb túlélése) felhasználásával végzik, vagyis ez azt jelenti, hogy minden generációval a legjobb elrendezéseket genetikailag irányítsák. ezáltal kialakítsák a következő generáció számára összeállított megoldást [3] [25].

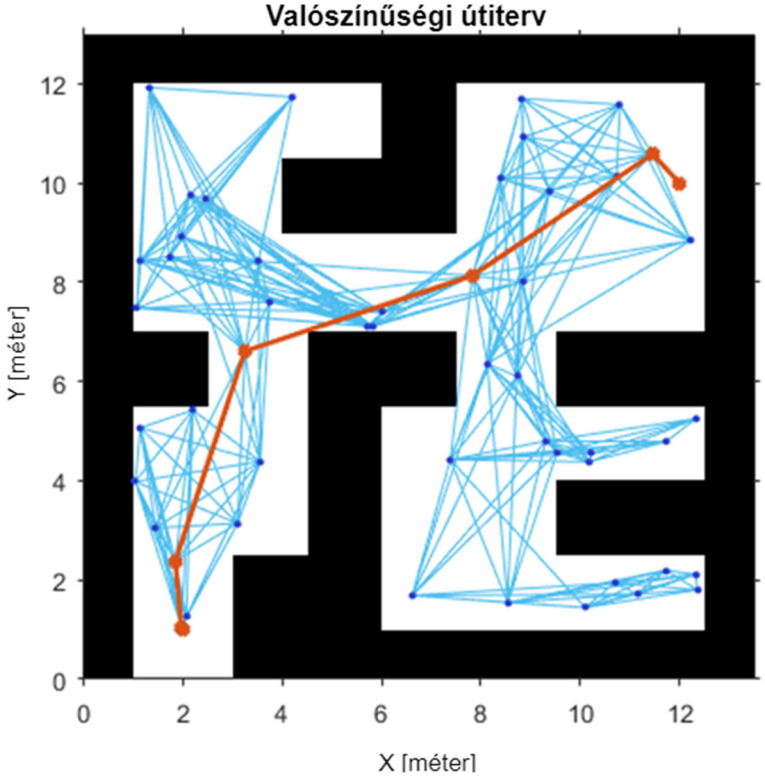
Az útvonaltervezés genetikus algoritmusának egyik példája egy olyan megközelítés, amely az útvonal távolságát és irányát használja a genetikai algoritmus genotípusaként [45]. A genetikai algoritmus folyamatán keresztül az út rövid távolsággal megvalósítható úttá válik. A genetikai algoritmus folyamatában a kezdeti megoldás génként jelenik meg. A genetikus algoritmus egyik folyamata a keresztezési folyamat. A keresztezési folyamat során az algoritmus véletlenszerű pontot választ és egy kromoszómával cseréli. Az átkelési folyamat

Szent István Campus, Gödöllő
Cím: 2100 Gödöllő, Páter Károly utca 1.
Tel.: +36-28/522-000
Honlap: https://godollo.uni-mate.hu

addig folytatódik, amíg az algoritmus meg nem találja a legjobb illeszkedést, amely a legrövidebb út és az ütközésmentes út. Általában a genetikai algoritmust használják optimalizálja az aktuális útvonalat. Han Baek szerint: "Az objektumok és a célpozíció megtalálása után egy genetikai keresési algoritmus aktiválódik az útvonaltervezőben, hogy via-pontokat generáljon a célhoz vezető rövid és biztonságos úthoz." Ezért a genetikai algoritmus optimalizálja az útvonaltervezőt [47].

Egy másik megközelítést mintavételen alapuló algoritmusnak neveznek. A mintavételen alapuló algoritmus hatékony egy nagy osztályproblémára, például a robotikára, a gyártásra és a számítógépes biológiára. A mintavétel alapú algoritmus egy magas dimenziós teret kezel. A klasszikus rács alapú módszerek, mint például az A* algoritmus optimálisan használhatók alacsony dimenziós térben [17], [28].

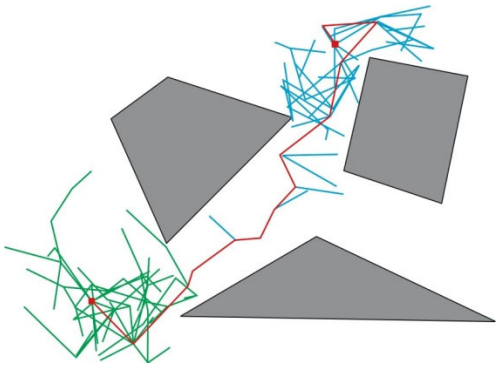
Az egyik mintavételen alapuló módszer a PRM (valószínűségi ütemterv). A valószínűségi ütemtervet a [27] ismerteti. A csökkent mozgásképességű pontok egy tanulási fázison mennek keresztül. A tanulási fázisban két lépés van, egy építési lépés és egy bővítési lépés. Az építési lépés az, ahol a mozgáskorlátozott személyek kezdetben véletlenszerű csomópontot hoznak létre egy térben. Ezután az algoritmus megpróbál csatlakozni az egyes csomópontokhoz egy másik csomóponthoz. A folyamat során egy helyi útvonaltervezőt hívnak meg. A helyi útvonaltervező létrehoz egy útvonalat egy csomóponttól egy csomópontig. Az egymáshoz közeli csomópontok csatlakoznak. Az a csomópont, amely már csatlakozott a másik csomóponthoz, nem lesz újra feldolgozva. A bővítési lépés az, ha egy csomópontnak nagy rése van a többi csomópont között, vagy van egy kis rés két akadály között, a csomópont közelében létrejön egy új csomópont-konfiguráció. Az új csomópont feldolgozása az építési folyamaton keresztül történik. A rendszer a teljes csatlakoztatott csomópontot lekérdezi. Ez azt jelenti, hogy az algoritmus megtalálja a csatlakoztatott csomópontokat a kiindulási pozíciótól a célpozícióig [38]. A mozgáskorlátozott pontok eredményét a 10. ábra mutatja [30].



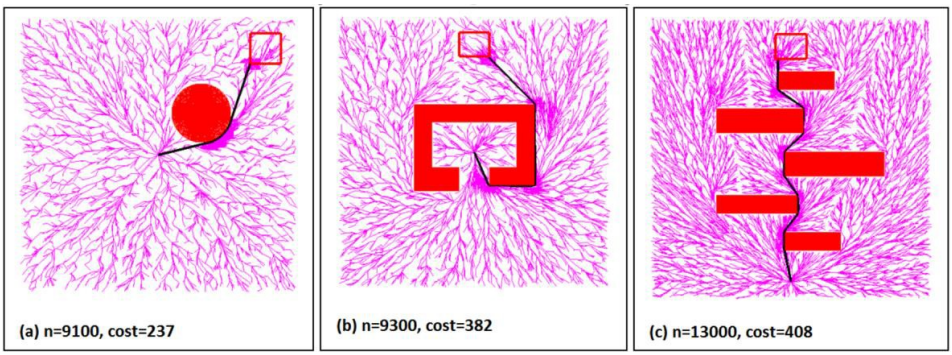
10. ábra: Valószínűségi ütemterv (PRM) eredménye [30]

Az útvonaltervezésben alkalmazott algoritmus újabb fejlesztése az RRT (Rapidly-Exploring Random Trees) néven ismert. Az egyszerű és legrövidebb útvonal újra megköveteli a kiegészítő algoritmust az RRT által nyújtott megoldás optimalizálására. Az RRT algoritmus egy olyan megközelítés, amely mintavételen alapuló tervezőket használt. A tervező két véletlenszerűen mintavételezett pontot köt össze egy térben. E pontok összekapcsolása egy olyan fát konstruál, amely a teljes teret feltárja [42]. Gyakran előfordul, hogy az útvonal gyorsan megtalálható a mintavétel-alapú tervezők használatával. A megoldás javításához további időre van szükség. Az RRT online útvonaltervezőként is ismert. Ezen algoritmusok inkrementális alkalmazásával elkerülhető a minták számának beállítása. Maga az RRT korlátlanul végtelen számú mintát képes generálni. Ennek köszönhetően az RRT algoritmus online problémákban is megvalósítható [22]. Ferguson és Stentz által készített kutatási cikke szerint az RRT algoritmus megoldása javítható az algoritmus többszöri futtatásával. Ők az első megoldást tekintették referenciának a következő megoldáshoz. [10]. Az RRT algoritmus által mintavételezett véletlenszerű pontok miatt cikkcakkos útvonalak jelenhetnek meg. Ennek a problémának a megoldására Karaman és Frazozoli az RRT* néven ismert módszert javasolta [42]. Az RRT* javítja az útvonal minőségét a fa újrahuzalozásával és a szomszédos csomópontok keresésével. Az

RRT és az RRT* összehasonlítása a 11. [19] és 12. ábrán látható [14].



11. ábra: RRT algoritmus eredménye [19]



12. ábra: RRT algoritmus eredménye [14]*

Azonban az RRT* algoritmusnak kompromisszumot kell kötnie az eredmény és a számítási idő között. Az RRT* algoritmusnak több időre van szüksége a sima útvonal előállításához [20].

2.3 Robot operációs rendszer

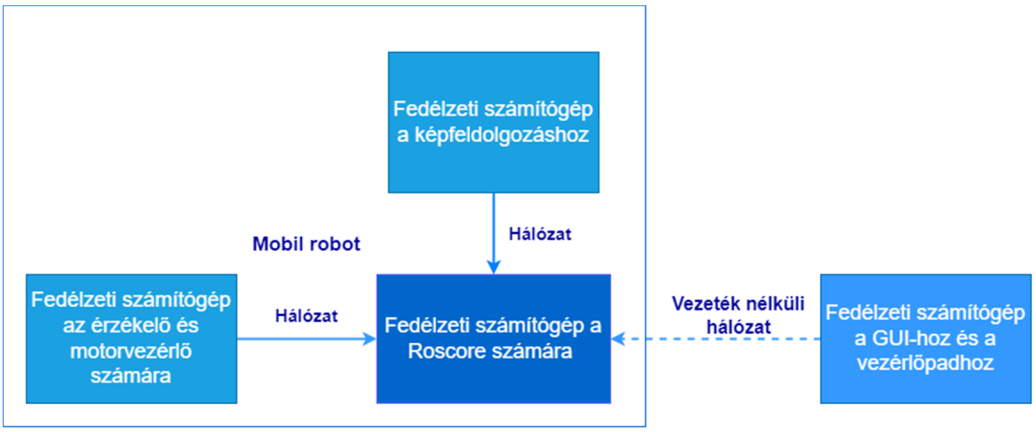
Robotika méretének és ebből kifolyólag hatókörének állandó fejlődése miatt, egyre nehezebb feladattá válik a robotok szoftverfejlesztése, hiszen minden robottípus más-más hardvert alkalmaz, így minden hardvernek pedig külön saját interfésze van. Például egy mobil robot két hardvert tartalmaz, az A hardver régi interfésszel van telepítve, addig a B hardver interfésze már új interfésszel használható, ebből következik a két interfész között nincs kompatibilitás [33].

A Robot Operating System (ROS) egy nyílt forráskódú szoftverkeretrendszer robotok részére és 2007 óta fejlesztették a Stanford Mesterséges Intelligencia Laboratóriumban. A robotok szoftvereinek fejlesztéséhez hozzájárul a ROS számos eszköze és könyvtára. A ROS első

verzióját Box Turtle-nak nevezték el. A Box Turtle verzió képes volt alacsony és magas szintű feladatokat végrehajtani, mint például érzékelők hozzáférését, diagnosztikai jelentéseket, autonóm navigációt, energiagazdálkodást, 1D és 3-D percepciót, vizualizációt é fegyvervezérlőket és is. A ROS három fő jellemzője myílt forráskód a peer-to-peer kapcsolat, és a többnyelvűség. [40].

2.3.1. Ügyfél-kiszolgáló kapcsolat

A ROS segítségével felépített rendszer olyan hosztokból áll, amelyek kliens-kiszolgáló kapcsolat keretében kapcsolódnak egymáshoz egy szerverrel. A szervert roscore-nak nevezzük. Ha a roscore leáll, akkor az összes kapcsolat megszakad, és az összes folyamat leáll. Például egy mobil robotot úgy terveztek, hogy képes legyen manuálisan és önállóan is mozogni. A mobil robotot egy vezérlő segítségével lehet irányítani. és egy kamera van csatlakoztatva, így a felhasználó a kamera képét látva láthatja a robot irányát. Az ROS hálózati kapcsolatot a 13. ábra mutatja [40].



13. ábra: Egy egyszerű ROS hálózati kapcsolat [40]

A mobil robotban lévő fedélzeti számítógépek Ethernet-en keresztül kapcsolódnak egymáshoz. A fedélzeti számítógép vezeték nélkül kapcsolódik a roscore-hoz. Ebben az egyszerű ROS hálózati kapcsolatban a mobil robotot a felhasználó vezeték nélkül vezérelheti. A probléma akkor jelentkezik, ha a vezeték nélküli kapcsolat nem stabil. Ebben az esetben egy erős vezeték nélküli routerre van szükség a kapcsolat stabilitásának fenntartásához. Így a fedélzeti számítógép által küldött adatok nem késnek. [2].

Minden állomásnak más-más folyamata van. A ROS-ban a folyamatot csomópontnak nevezik. A csomópont lehet bármilyen folyamat, például érzékelő mérés, adatfúzió, vezérlő és

egyéb folyamatok. Az egyes csomópontok üzenetek közzétételével és feliratkozásával kommunikálhatnak egymással. Ez az üzenet a csomópont által előállított adatokat tartalmazza. Minden üzenetnek saját neve van. A közzétételi folyamat az, amikor a csomópont az adatokat a kiszolgálónak továbbítja. Bármelyik csomópont, amelyik csatlakozik a kiszolgálóhoz és feliratkozik a megfelelő nevű üzenetre, felhasználhatja az adatokat. [29]. Például van egy beszélő csomópont és egy figyelő csomópont. A beszélő csomópont közzétesz egy "chatter" nevű sztringadatot a hálózaton. A figyelőcsomópont feliratkozik a "chatter" nevű sztringadatokra. A ROS-ban van egy eszköz a ROS rendszer grafikonjának nyomtatásához, ezt rqt_graph-nek hívják. A példa ROS rendszergrafikonja a 14 ábrán látható [29].

14. ábra: A ROS rendszer grafikonja [29]

2.3.2. Többnyelvű programozás

A ROS jelenleg négy különböző programozási nyelvet támogat, a C++, az Octave-t a Python-t és a LISP-t, a felhasználók által elmondható, hogy a ROS legnépszerűbb programozási nyelvei a C ++ és a Python.

A csomópont C++, Python, Octave vagy LISP nyelven írható. A következő kérdés áll fenn, hogy honnan tudhatja egy csomópont, hogy egy másik csomópont által a fogadott üzent érvényes adat-e és azonos adattípust használ-e? A kérdések megválaszolásra a ROS egy egyszerű nyelvsemleges interfész definíciós nyelvet (IDL) alkalmaz a csomópontok közötti adat cseréhez. Egy rövid szövegesfájlt használ az IDL az üzenetek pontosításához. A ROS keretrendszer kibővíti a IDL-t minden programozási nyelvre, így minden programnyelvnek megvan a megvan az egyéni üzenetdefiníciója a nyelvével. [34].

A ROS egy nyílt forráskódú keretrendszer, ezáltal a nyilvánosan elérhető a forráskódja. A ROS a BSD (Berkeley Software Distribution) licencen alapszik, így elérhetővé teszi a kereskedelmi és nem kereskedelmi projektek létrehozását, fejlesztését. Bárki szabadon használhatja a ROS-t, mivel nem igényel modult, mivel az adat, folyamatok közötti kommunikáción keresztül megy át a modulok között [7].

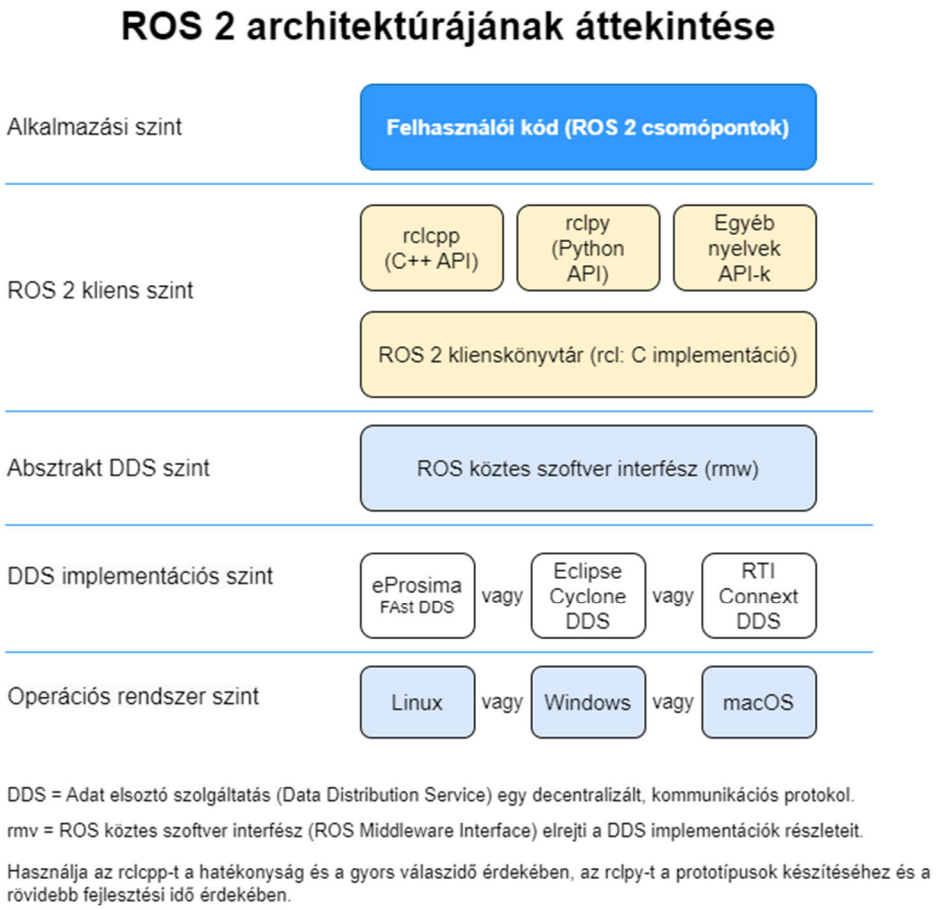
Szent István Campus, Gödöllő
Cím: 2100 Gödöllő, Páter Károly utca 1.
Tel.: +36-28/522-000
Honlap: https://godollo.uni-mate.hu

2.3.3. A ROS 2 architektúrája és főbb jellemzői

A ROS 2, a ROS továbbfejlesztése, merthogy a ROS1 már nem támogatott, nem fejlesztik, a korábbi ROS-verziótól egy többrétegű architektúrát örökölt, amint azt a 15. ábra mutatja. Az alkalmazási réteg az egyes programozási nyelvekre jellemző kliens könyvtárak halmaza. A fő támogatott nyelvek a Python és a C++. Az alkalmazásprogramozási interfész (API) alkalmazásával stabilitást biztosítson a különböző programozási nyelveken írt programok számára. Az adatelosztó szolgáltatás (DDS) egy valós idejű kommunikációs ipari szabvány, ez helyettesíti a roscore-t. A ROS 2 úgy valósítja meg, hogy megfeleljen a valós idejű követelményeknek, miközben fenntartja a kiadó/kiíró elvet [50]. A DDS és az rcl közötti kommunikáció megkönnyítése érdekében a ROS egy middleware könyvtárat használ: "rmw" (ROS middleware library). A DDS szállítási módszere tökéletesen illeszkedik a ROS víziójába, mivelhogy bevezet egy publishert és egy subscribert. A DDS az "Interface Description Language (IDL)" nyelvet használja, amelyet az Object Management Group (OMG) meghatározott az üzenetek definiálására és szerializálására. DDS által kínált transzportot az alapértelmezett felfedező rendszerrel kell használni, amely egy elosztott felfedező rendszer. Ezáltal megszünteti a ROS masterhez hasonló berendezés szükségszerűségét és engedélyezi a kommunikációt bármely két DDS program között, ebből fakadóan rugalmasságot ad hozzá és megvédi a rossz kapcsolati hálózatoktól. Nem szükséges a dinamikus felderítési folyamat alkalmazása, mivel számos DDS-szállító ajánl statikus felderítési eredményt. A ROS 2 rendelkezik számos kulcsfontosságú tulajdonsággal, ami által széles körben népszerűvé és elfogadottá tette ezt a keretrendszert, mint például tudományos, ipari és az akadémiai környezetben is. A ROS 2 felhasználók számára nem csak a nyílt forráskódú könyvtárak elérhetőek hanem azon kívül nagy teljesítményű eszközök széles skálájához is hozzáférnek, beleértve a szimulációs szoftvereket, az útvonaltervezést és a vizualizációt. Összességében elmondható ez a csomag nagy segítséget jelent a robotrendszerek tervezésében. Ezen túlmenően a ROS 2 további jellemzői a következők [12]:

- Felfedezés, szállítás és szerializálás az adatelosztási szolgáltatáson (DDS) keresztül
- Több csomópont koordinálásához fejlett indítórendszer
- A menedzselt életciklusú csomópontok támogatása
- Hozzájárulás különböző DDS implementációkhoz
- Közzététel/feliratkozás témákon keresztül

- A nem ideális hálózatok kezeléséhez szolgáltatásminőségi beállítások
- Közös alapvető ügyfélkönyvtár
- Valós idejű kód előzetes támogatása



15. ábra: A ROS 2 architektúra. [4]

2.4 Robot és környezeti modell

A szimuláció elsődleges feladata a robotmodellek tervezése. Egy robottervet alkottak meg, amelynek az első lépése a tervezés volt és CAD-ben (Computer Aided Design) hajtották végre és a dinamikus ízületekhez megfelelő vezérlőket hoztak létre. A két különálló fájlt használtak, az első fájt, a szimulációt Gazebóban dolgozták fel és a második fájlt pedig az univerzális robotleíró formátumban, az RVIZ-ban történt a megjelenítése [36]. Ezen túlmenően előállítottak egy új szimulációs szoftver robot alkalmazásfejlesztőt és üzemeltetési környezetet, amelyben beimportálták a modellt, így egyszerre jelent meg a Gazebo és a robot vizualizációja a képernyőn. A robot vizualizációs plugin segítségével kifejlesztettek egy grafikus felhasználói felületet, a GUI-t. A GUI segítségével a szimuláció nagyon könnyen

összehasonlítható. Jelenleg a szimulációs szoftverek széles körben elérhetők a piacon, de az alkalmazáshoz általánosan választott szoftver nyílt forráskódú, ezért előnyös a közös kapcsolat a Gazebo és ROS között. A szerző létrehozott egy szimulációt Gazebo-ban, amely a Linux operációs rendszer alatt fut és ROS keretrendszeren működik. A projektben az akadályok távolságának kiszámítását elvégezték ultrahangos érzékelők segítségével, amelyek már a robottervezés során rögzítve vannak. Az érzékelőktől összegyűjtött adatokat az RVIZ adatgyűjtő szoftvert alkalmazták, valamint további összehasonlítási lépésekhez használták [32].

Számos bővítményt tartalmaz a Gazebo. Az egységesített robotikai leíró formátum (URDF) egy bővíthető jelölőnyelvi (XML) fájltípus, amely tartalmazza a robot fizikai leírását. A ROS-ban történő megjelenítésre használják a környezetek és a robotok a fájlt [51]. A szerző a robothoz számos funkciót adott hozzá, amelyek az akadályok felismerésére szolgáltak, például a 3D lézeres kereső, az IR érzékelő és a kamera. A fizikai tulajdonságokat, mint a tehetetlenség, a súrlódás szintén hozzákapcsolták a modellezéshez a Gazebóban a dedikált URDF fájlhoz [26].

A szerzők bemutatták robot modellezését és azt, hogy hogyan importálták a modellt elemzésre. Az URDF meglehetősen jó és pontos illetve előnye még, hogy a modell elemzése valamint a és nyílt forráskódú szoftverbe való importálása egyszerű [26], [32], [36], [51].

2.5 A szakirodalmi áttekintés következtetései

Az autonóm mobil robotok napról napra frissülnek új funkciókkal. A korlátozások egyre kisebbek, mielőtt A ROS fő követelménye a Linux operációs rendszer volt, viszont az újfejlesztésnek köszönhetően már Windows oprendszerben is futtatható a ROS2 kerretrendszer, tehát a hozzáférhetőség egyre szélesebb. A ROS 2-vel való integráció megkönnyíti és kényelmessé teszi a szimulációs megvalósítást. A különböző képfeldolgozó algoritmusok és vizuális érzékelők alkalmazása megbízhatóbbá teszi a leképezési technikát. A mozgórobot szimuláció a Gazebo-ban is elvégezhető a modell URDF-ben történő tervezése után. Az algoritmusok eszközei és típusai életképes lehetőséget jelentenek a szimulációk megvalósítására és fejlesztésére.

Szent István Campus, Gödöllő
Cím: 2100 Gödöllő, Páter Károly utca 1.
Tel.: +36-28/522-000
Honlap: https://godollo.uni-mate.hu

3. Alkalmazott módszerek

Ebben a fejezetben bemutatom a projekthez alkalmazott különböző szoftvereket és fájl rendszereket. A feladat során használt programozási nyelvek is ebben a szakaszban kerülnek részletezésre. Az szakirodalom feldolgozás fejezetben említett eszközök és algoritmusok működőképes lehetőséget biztosítanak a szimulációk létrehozásához és továbbfejlesztéséhez.

3.1 A feladat fejlesztési rendszerének beállítása

Az Ubuntu 20.04.2. LTS operációs rendszert alkalmaztam, a mozgórobot szimulációs projekt alkalmazásához. Az Ubuntu e verziója hosszú távú támogatást nyújt és kompatibilis a projekthez használt legtöbb csomaggal és fejlesztőeszközzel. Továbbá, ez a szoftver ingyenes és nyílt forráskódú, amely letölthető az Ubuntu hivatalos weboldaláról.

3.2 A Gazebo rövid bemutatása

A Gazebo, a robotikai iparban az egyik használt szimulátora. A Gazebo egy 3D szimulátor, amely képes komplex beltéri és kültéri környezetekben robotpopulációkat pontosan és hatékonyan szimulálni. Jobb szimulációs eredményeket ad és széles körű érzékelőket és interfészeket kínál mind a felhasználók, mind a programok számára. Az adatvizualizáció az RVIZ eszközzel, a komplex környezetek szimulációja és a fordított tervezés azok a fő jellemzők, amelyek a Gazebo-t más szimulációs szoftverektől megkülönböztetik. A Gazebo fő elemei a következők:

A, Világfájlok

Egy háttér, ami a kamera pozíciójának és a szimulációs lépésméret szimulációjára használható.

- Autonóm navigációs világ

B. Modellek

Számos modellt található a Gazebo könyvtárában, ezek a modellek jól felépítettek és importálhatók a világba a további szimuláció célkitűzésként

C. Gzserver

A gzserver feladata az előbb említett világfájl beolvasása, hogy létrehozzon és betöltse a világot. A gzserver indítja el a fizikai frissítési ciklust és a szenzoradatok generálását. A gzserver a Gazebo alapvető alkotórésze és grafikus felülettől függetlenül is használható.

Szent István Campus, Gödöllő
Cím: 2100 Gödöllő, Páter Károly utca 1.
Tel.: +36-28/522-000
Honlap: https://godollo.uni-mate.hu

3.2.1 A Gazebo bővítmények áttekintése

A plugin egy olyan kódrészlet, amelyet egy megosztott könyvtárba fordítanak, majd a szimuláció során használnak. A C++ nyelven keresztül a plugin közvetlen hozzáférést biztosít a Gazebo minden funkciójához. A pluginok fő felhasználási területeit az alábbiakban ismertetjük:

- Önálló rutinok, amelyek könnyedén megoszthatók;
- A fejlesztők képesek irányítani a Gazebo szinte minden aspektusát;
- Lehetőség van arra, hogy egy működő rendszert installáljanak és eltávolítsanak;
- Lehetőség van a szimuláció programozott módosítására, mint például modellek beillesztés, mozgatása az eseményekre való reagálás

3.3. Egységes robot leíró fájl

Az egységes robot leíró formátum (Unified Robot Description Format, URDF) a ROS 2 által használt formátum a robotmodellek tárolására, valamint a ROS2 szimulációs formátuma. Az URDF valójában a ROS központi eleme és számos csomagja használja. Lehetséges a ROS használata URDF nélkül is, de ebben az esetben sok része nem fog megfelelően vagy egyáltalán nem fog működni. Technikailag az URDF egy XML-alapú tartományspecifikus modellezési nyelv, amely lehetővé teszi a kinematika és a dinamika bizonyos részeinek valamint a robothoz kapcsolódó metaadatok különböző más részeinek kódolását olyan formátumban, amely bizonyos mértékig olvasható és írható mind az emberek, mind a gépek számára. Más szavakkal, az URDF az a fájlformátum, amelyet a ROS2 használ a robot testének elrendezésének leírására, beleértve annak összes kapcsolatát, ízületét, formáját és színét. Lehetővé teszi továbbá, hogy olyan további információkat tároljunk, mint például a robot összes ízületének mozgástartománya és az ezeket a mozgásokat végrehajtó sebesség. Ez egy szakterület-specifikus nyelv, mivel a robotika és a robotmodellezés területére jellemző terminológiát használ. Ez lehetővé teszi, hogy gyorsan létrehozzunk robotmodelleket a robot részeinek valós neveit használva, ahelyett, hogy általános leírásokhoz kellene fordulnunk.

Egy URDF fájl csupán egy szöveges fájl, amely XML címkéket tartalmaz: olyan speciális kulcsszavakat, amelyeket a ROS 2 az URDF részeként azonosít. Ezen XML címkék közül néhány más fájlokra vagy a robotok részeinek 3D modelljeire hivatkozik, ami lehetővé teszi számunkra, hogy külső fájlokat használjunk, hogy gyorsan beépítsünk egy

Szent István Campus, Gödöllő
Cím: 2100 Gödöllő, Páter Károly utca 1.
Tel.: +36-28/522-000
Honlap: https://godollo.uni-mate.hu

részletes, színekkel ellátott alakzatot egy 3D hálós fájlból, például egy felső vagy alsó kart. Az URDF fájl nagy részét a linkeknek és ízületeknek nevezett elemek teszik ki. Ezek közvetlenül megfelelnek egy robot linkjeinek és ízületeinek: a linkek adják a robotok formáját, az ízületek pedig összekötik a linkeket, és meghatározzák, hogyan mozoghatnak egymáshoz képest. Tehát egy robot az URDF-ben egy bizonyos sorrendben csuklókkal összekapcsolt linkek gyűjteménye. Azonban ha ez a sorrend megváltozik, akkor a robot által végezhető mozgások is megváltoznak, tehát nem csak a test elrendezése változik a linkeket összekötő ízületek módjának megváltoztatásával, hanem a robot viselkedése is.

3.4 Szimulációs eszközök

Ebben az alfejezetben ismertettek néhány szimulációs eszközt, az itt leírt információk segítségével könnyebben értelmezhető a járőr robot navigációjának fejlesztése során alkalmazott lépések, parancsok, kifejezések.

3.4.1. tf

A tf (Transform) egy könyvtár, amely képes nyomon követni a robot összes koordinátakeretét a ROS keretrendszeren belül működik. Tehát ez nem központosított szolgáltatás, hanem egy szabványos protokoll, amely lehetővé teszi a csomópontokon keresztül, a transzformációs adatok elosztott rendszerben történő megosztását. A tf könyvtár számos felhasználási lehetőséget kínál a mozgórobot statikus és dinamikus jellemzőinek kezelésében. Mivel minden keret rendelkezik egy azonosítóval, így a többszörös transzformáció során nem történik adatvesztés, ezáltal az adatok könnyen nyomon követhetőek és egyszerűbb meghatározni, hogy melyik keret tartozik az adott adatokhoz. Az adatok korábbi helyzetére vonatkozó információkat is tárolnak és könnyen hozzáférhetőek, de ez a felvétel megkezdése előtt nem lehetséges.

- Segédmetódusok

  Ezek olyan metódusok, amelyek képesek fogadni a tf-adatokat, beolvasni a fejlécet az időbélyeg és a keretazonosító megszerzése érdekében, valamint alkalmazni a transzformációt a kapott adatokra. Ezeket az adattípusokat, üzeneteket és tf adattípusokat a ROS2 definiálja.

3.4.2. RVIZ

Egy adatgyűjtő eszközként működik, összegyűjti az érzékelők adatait és tárolja azokat

Szent István Campus, Gödöllő
Cím: 2100 Gödöllő, Páter Károly utca 1.
Tel.: +36-28/522-000
Honlap: https://godollo.uni-mate.hu

a későbbi elemzésekhez, tehát egy robotvizualizációs alkalmazás. Ez a 3D-s vizualizációs eszköz lehetővé teszi a robot aktuális környezetének és a szenzoradatok konfigurációs modelljének megtekintését. Az RVIZ-t használják az érzékelők, mint például a kódolók, kamerák, elmozdulásérzékelők, ultrahangos érzékelők és más útvonaltervezési érzékelők adatainak valós idejű mérésére és kvantifikálására. Az RVIZ ablakán keresztül 2D-s és 3D-s navigáció hajtható végre a modellen. Az RVIZ-ben az odometriát fogjuk tanulmányozni. Az RVIZ ideális eszköz arra, hogy meghatározzuk, mi a hiba egy látórendszerben. A bal oldali listán minden elemhez tartozik egy jelölőnégyzet, amellyel azonnal megjeleníthetünk vagy elrejthetünk néhány vizuális részletet.

3.4.3. Egyidejű Navigáció és Térképezés SLAM

Az Egyidejű Navigáció és Térképezés (SLAM, Simultaneous Localization and Mapping) az ROS és ROS 2 alkalmazásai közé vált. A SLAM célja a mobil robotikában, hogy a robot egy ismeretlen környezetben mozogjon, miközben helyi szenzorok segítségével folyamatosan ismereteket szerez a környezetről, végül létrehozva egy 2D/3D állandó térképet. A SLAM algoritmusok és a navigációs szoftver kombinációja létfontosságú, mivel egymást kiegészítik. A SLAM implementációk többsége szenzorinformációt igényel a térkép létrehozásához, majd később közzéteszi azt egy témán keresztül. A lokalizációhoz a robotnak következetes és pontos térképre van szüksége; egy térkép létrehozásához a robotnak jó becslésre van szüksége a helyzetéről. Ez erősen összekapcsolja a SLAM algoritmust a navigációs térrel. Egy navigációs szoftverrel a robot megváltoztatja pozícióját, így mozgás közben a mozgórobot folymatosan felfedezi a környezetét. Az ilyen információkat továbbítják a SLAM szoftvernek, amely térképezi a becsült jelenlegi helyzetet az észlelt tárgyakkal kapcsolatos adatokkal, ezáltal létrehozva egy térképet (vagy annak részeit), amit a navigációs szoftver felhasznál.

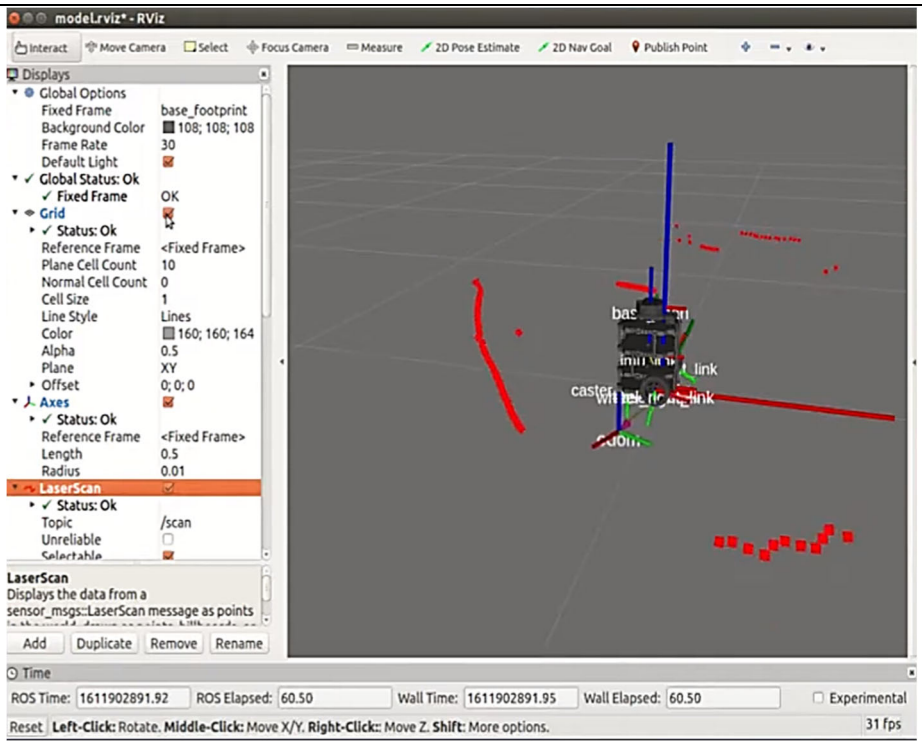
Ebben a munkában az ROS gmapping és az ROS 2 SLAM Toolbox csomagokat használtam. A Gmapping egy egyszerű lézer-alapú SLAM megoldást kínál egyetlen csomóponton keresztül. A Gmapping lehetővé teszi egy 2D elfoglaltsági rács térkép létrehozását. A SLAM Toolbox három fő működési módot és futtatható állományt kínál: szinkronizált térképezés, aszinkron térképezés és tiszta lokalizáció. A SLAM kihívás segítése érdekében a szinkronizált térképezés lehetőséget kínál a térképezésre és pozicionálásra egy térben, miközben megőrzi a megfigyelések hátralékát. Ez előnyös lehet az offline feldolgozás során, vagy amikor a térkép minősége különösen fontos. Az

Szent István Campus, Gödöllő
Cím: 2100 Gödöllő, Páter Károly utca 1.
Tel.: +36-28/522-000
Honlap: https://godollo.uni-mate.hu

aszinkron mód ezzel szemben csak új méréseket dolgozz fel, miután az előző vizsgálat befejeződött és az újonnan frissített követelmények teljesültek. Ennek eredményeként ez soha nem marad le valós időben, miközben bonyolult hurokzárásokat hajt végre. Mindazonáltal, ha a végső értékelés feldolgozása túl sokáig tart, a térkép lehet, hogy nem tartalmazza az összes jogos mérést. Amikor a valós idejű lokalizáció pontossága különösen fontos, ez az opció előnyös. A tiszta lokalizációs módot nem lehet használni a környezeti változások rögzítésére. Ehelyett összehasonlítja az aktuális munkamenet során gyűjtött adatokat az eredeti munkamenet(ek) méréseivel és pozíció-gráfjával egy folyamatos mérési puffer segítségével.

3.5. Odometria

Az emberek rendelkeznek kiváló látási képességgel és memóriával, amelyek segítségével tudják, hol tartózkodnak, a robotok esetében sincs másképp, mert a robot pozíciójának ismerete rendkívül fontos. Az odometriának nevezzük, a robot saját helyzetének meghatározását a környezetben. Ez folyamat, a mozgásérzékelők alkalmazását jelenti a robot helyzetének változásának nyomon követésére egy ismert pozícióhoz képest. A megtett távolság a kérékfordulatok számításával meghatározható, ha a mozgórobot egyenes vonalban halad és már ismeri a kerekeinek átmérőjét. Az odometriát általában mobil robotok érzékelőjeként használjuk, mivel minden mobilrobot képes érzékelni a kerekek forgását. Azonban az odometria hibái idővel felhalmozódhatnak a kerekek csúszása vagy a numerikus integrációs hiba miatt. Emiatt az odometriát gyakran kiegészítik külső érzékelőkkel, mint például kamerák, távolságérzékelők és GPS-t használó becslési technikák. A mozgórobot esetében ez egy három Omni kerékkel felszerelt robot, differenciális meghajtással, amely két független motorral rendelkezik a szabad mozgás érdekében. A 16. ábra bemutatja a robot odometriájának RVIZ vizualizációját mozgás közben.



16. ábra: Az Odometria részletei és a lézerszken funkció az RVIZ vizualizáción keresztül

4. Autonóm navigáció beállítása

A projekt célkitűzése egy járőr robot létrehozása a ROS 2 és a Gazebo szimuláció felhasználásával. A feladatban négy pontot határozok meg a robot navigálásához, így a robot folyamatosan mozog az egyik ponttól a másikig, majd újra és újra megismétli ezt a navigációs folyamatot, tehát állandó mozgást végez a négy pont között. Egy saját alkalmazási réteget készítek, ahol egy külső viselkedés fa segítségével utasítom a navigációs csomagot, hogy a meghatározott négy pont valamelyikéhez irányítsa a járőr robotomat. A viselkedés fát C++ programoztam.

4.1. ROS 2 alapú navigáció beállítása szimulációval

A projekt megkezdése előtt két fő navigációs komponensről kell beszélnem a mozgórobot szimulációja kapcsán, az egyik a lokalizáció, a másik pedig a térképezés létrehozása. A lokalizáció esetében a TurtleBot3 szimuláció könnyen létrehozható Gazebo-ban, mivel az odometria jó értékeket vesz fel. A második részt, ami a térképezés, meg kell alkotnom a szimulációhoz. Először létrehozok egy térképet, majd használom a navigációs csomagokat a mozgórobot koordinálásához. Technikailag erre van szüksége a robot agyának ahhoz, hogy feldolgozza a kapott információkat és elhatározza, hogy elinduljon egy sík területen. Ebben az esetben a robotunk agya a navigációs csomag.

Összegezve az elmondottokat, meghatározom a lokalizációt és a térképezést, vagyis a navigációhoz szükséges két információt, mivel A Nav2-nek szüksége van a robot aktuális helyzetére és a térképére. A Nav2 bemenetei adatai a két TF, transzformáció .,A globális helymeghatározó rendszer (GPS, SLAM, Motion Capture) feladata, hogy legalább az *map->odom* parancs átalakítást végezze el Az útmérő rendszer feladata, az *odom-> base_link* transzformáció biztosítása. Az útmérő számos forrásból származhat, beleértve a LIDAR-t, a RADAR-t, a kerékkódolókat, a vizuális-nézeti odometriát (Visual-Inerial Odometry, VIO) és az inerciális mérőegységet (Inertial Measurement Unit, IMU) is. Az odometria célja, hogy a robot mozgásán alapuló sima és folyamatos lokális keretet biztosítsa. A globális helymeghatározó rendszer frissíti az átalakítást a globális kerethez képest, hogy figyelembe vegye a kilométer-eltolódást. Különféle típusú érzékelőket fogad be *N* folyamatos és egyenletes kilométer-mérést biztosít a TF-hez és egy adott témához. Egy tipikus mobil robotikai rendszerben előfordulhat, hogy a kerékkódolók, az IMU-k és a látás odometriája ilyen módon egyesül. A sima kimenet ezután felhasználható a precíz mozgáshoz és a robot

pozíciójának pontos frissítéséhez, a globális pozíciófrissítések között. A térképezésnél, a SLAM-et igénylő alkalmazásokban a navigációval párhuzamosan történik. Jelenleg azonban egy statikus térképet alkalmaztam a navigációhoz. A ROS 2-ben a SLAM Toolbox segítségével, statikus térképet állítottam elő és mentettem a navigációhoz.

A, Lokalizáció beállítsa:

Először feltelepítettem a ROS 2 keretrendszeremet, utána elindítottam egy munkaterületet, terminált, a TurtleBot3 számára, beimportáltam az összes TurtleBot csomagot az SRC könyvtárba. Ehhez készítettem a VCS eszközt, majd létrehoztam egy szövegfájlt és a VCS eszközt használom az adatok importálására. Itt található az TurtleBot3 repók nevei, amit használni fogok, a terminálba megírt parancsokat a 17.ábra szemlélteti.

```
lehoczki@lehoczki-VirtualBox:~$ mkdir -p turtlebot3_ws/src
lehoczki@lehoczki-VirtualBox:~$ cd turtlebot3_ws/src
lehoczki@lehoczki-VirtualBox:~/turtlebot3_ws/src$ gedit turtlebot3.repos
lehoczki@lehoczki-VirtualBox:~/turtlebot3_ws/src$ vcs import . < turtlebot3.repos
.....
=== ./turtlebot3/turtlebot3 (git) ===
Cloning into '.'...
=== ./turtlebot3/turtlebot3_msgs (git) ===
Cloning into '.'...
=== ./turtlebot3/turtlebot3_simulations (git) ===
Cloning into '.'...
=== ./utils/DynamixelSDK (git) ===
Cloning into '.'...
=== ./utils/hls_lfcd_lds_driver (git) ===
```

17. ábra: A megírt parancsok a terminálban

A munkaterületemet a *colcon build - -symlink -install* paranccsal elkezdtem felépíteni, fontos megjegyezni, hogy ezt ne az SRC könyvtáradból tettem meg, hanem visszatérek a munkaterületre és elindítottam a "*colcon build*" parancsot, ezután mind a 16 csomag feltelepült, a folyamatot a 18 ábra mutatja be.

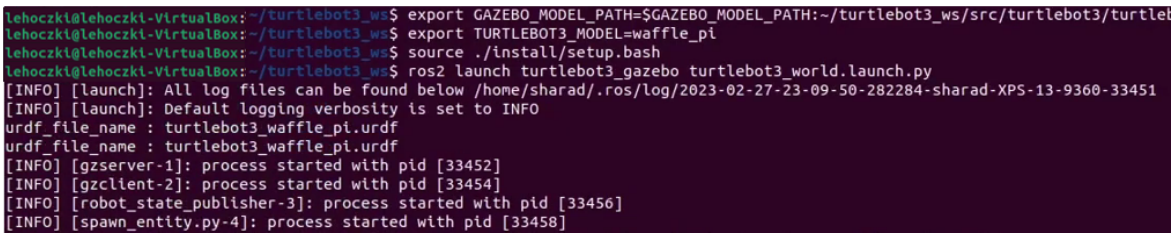
```
lehoczki@lehoczki-VirtualBox:~/turtlebot3_ws/src$ cd ..
lehoczki@lehoczki-VirtualBox:~/turtlebot3_ws$ colcon build --symlink-install
Starting >>> turtlebot3_msgs
Starting >>> dynamixel_sdk
Starting >>> hls_lfcd_lds_driver
Starting >>> turtlebot3_description
Finished <<< turtlebot3_description [3.77s]
Starting >>> dynamixel_sdk_custom_interfaces
Finished <<< dynamixel_sdk [5.60s]
Starting >>> turtlebot3_cartographer
Finished <<< turtlebot3_cartographer [4.99s]
Starting >>> turtlebot3_gazebo
Finished <<< hls_lfcd_lds_driver [28.7s]
Starting >>> turtlebot3_navigation2
[32.1s] [4/16 complete] [4 ongoing] [turtlebot3_msgs:build 59% - 31.8s] [dynamixel_sdk_custom_interfaces:build 71% - 28.0s]

Finished <<< turtlebot3_node [2min 0s]
Starting >>> turtlebot3_bringup
Finished <<< turtlebot3_bringup [1.41s]
Starting >>> turtlebot3
Finished <<< turtlebot3 [1.34s]

Summary: 16 packages finished [2min 58s]
  3 packages had stderr output: turtlebot3_example turtlebot3_gazebo turtlebot3_teleop
```

18. ábra: A csomagok feltelepítése

A szimuláció elindításhoz, így minden készen áll a Gazebo-ban, először be állítottam a Gazebo modell elérési útvonalát a TurtleBot számára. Ehhez futtattam az "export *GAZEBO_MODEL_PATH*" parancsot a TurtleBot modelljének exportálásához majd utána elindítottam a mozgórobot modell a Gazebo szimulációs környezetben ennek a terminál forrása a ROS 2 számára a következő parancs: *source /opt/ros/galactic/setup.bash* ha ezt elvégeztem indítom a szimulációt a következő paranccsal *ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py* a művelet lépéseit 19. ábra prezentálja. A szimulációs rész ezzel elkészült, így átértem a térképezés létrehozásához.

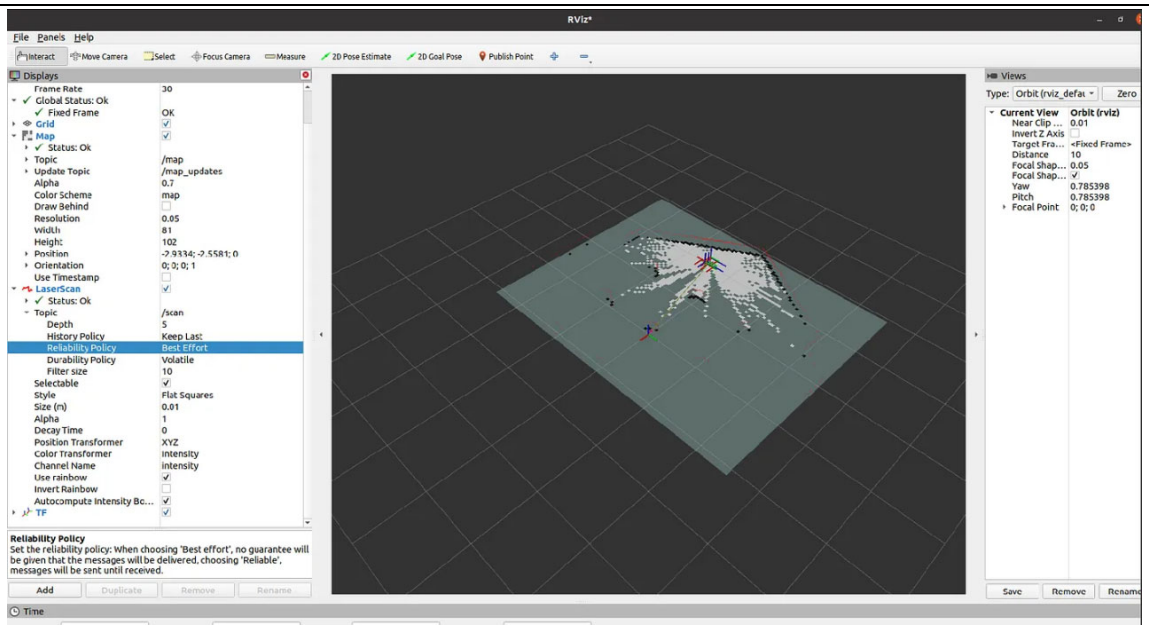


19. ábra: A Gazebo elindításhoz szükséges parancsok

Ahhoz, hogy térképet készítsek, egy új terminált nyitok és elindítom a "ros2 launch slam_toolbox_online_async_launch.py" parancsot. Ez a futtatás segít nekem a térkép készítésében. Az odom egy téma, ami megadja nekünk az odometria értékeket és odom->base_link a transzformációt. Ha valódi robotot használok, akkor ezek az útmérő értékek zajosak és hajlamosak az elsodródásra. Tehát összekapcsoltam őket más szenzorkimenetekkel, például az IMU-val (helyi) és a GPS-szel (globális) a Robot Localization Package segítségével, így számomra a lokalizációs értékek készen állnak.

B, A statikus térkép beállítása:

A térkép készítéséhez, egy új terminált nyitottam meg és ehhez a SLAM eszköztárat használom a ROS 2-ben és a Slam toolbox csomagot futattam a térképezéshez. Az új terminálba a két parancsot elindítottam, az egyik a *source /opt/ros/galactic/setup.bash,* a másik pedig a *ros2 launch slam_toolbox online_async_launch.py*, ezekután aktiváltam a RViz-t a szükséges vizualizációkkal. Egy új munkaterület megnyitását követően elindítottam az RViz-t, a *source /opt/ros/galactic/setup.bash rviz* kóddal majd a megjelenítéshez, az RViz felületét a 20. ábra ismerteti és a következő konfigurációkat adtam hozzá: a lézerszkennelést (téma /scan), a megbízhatósági szabályzatot (Best Effort), a térképet (téma /map) és a tf-et.

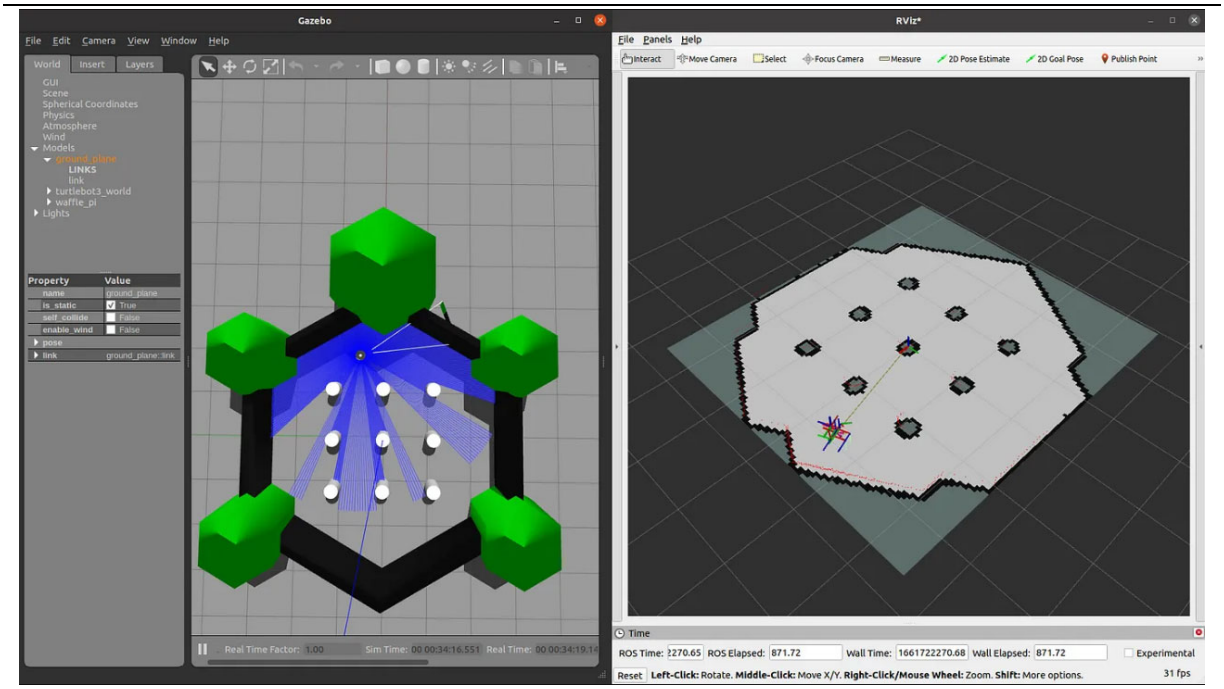


20. ábra: Az RViz felülete

Teleop futtatása a robot mozgatásához és térképezéséhez volt szükéges, melyet egy új terminál ablakban a következő parancsok indításával végztem el:

- source /opt/ros/galactic/setup.bash
- ros2 run teleop_twist_keyboard teleop_twist_keyboard

Egymás mellé helyeztem a Gazebo szimulációs és a RViz ablakát, hogy lássam az RVizben készülő térképet, a teleopálási a folyamatot a 21. ábra szemlélteti. Ahhoz, hogy mindent vizualizálni tudjak, elkezdtem irányítani, mozgatni a robotot, mivel a robot kamerája látja a világ különböző területeit, ezt a műveletet addig végeztem, míg fel nem épült a teljes térkép.



21. ábra: A mozgórobot Gazeboba történő mozgatással az RVizbe elkészül a teljes térkép

Egy új terminálban aktiváltam a *nav2_map_serve*-t, a következő két parancssal:

- source /opt/ros/galactic/setup.bash
- ros2 run nav2_map_server map_saver_cli -f ~/map

Ezekkel a modulokkal a térképet a kezdőkönyvtáromba menti a *map.pgm* és *map.yaml* formátumként, ezzel befejeztem a térkép létrehozásának folyamatát.

A navigáció létrehozása

Az elkészített lokalizáció és térkép után, elkezdtem a navigációm felépítését. A navigációhoz a "nav2_bringup" csomagot használtam, így ismét alkalmaztam egy új terminált és futtattam a "*nav2_bringup*" alkalmazást. Ez csomag a navigáció összes szükséges komponensét elindította, beleértve az összes navigációs és visszaállító viselkedést is, amelyek az alábbi modulok:

- source /opt/ros/galactic/setup.bash
- ros2 launch nav2_bringup bringup_launch.py use_sim_time:=True
- autostart:=True map:=/path/to/your-map.yaml

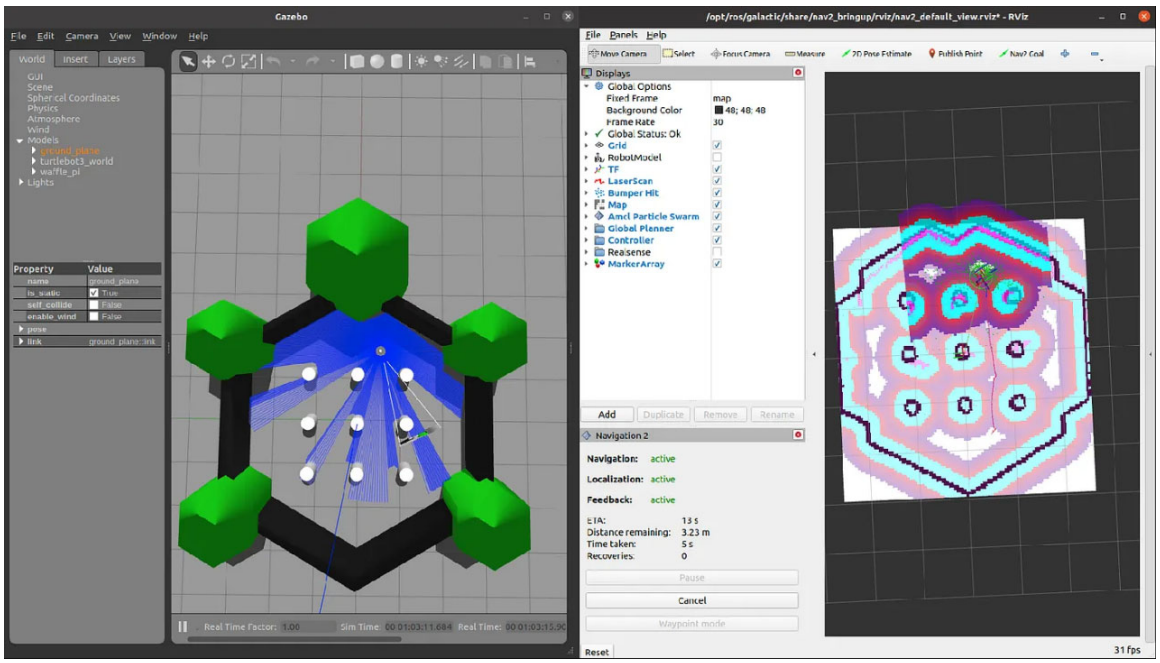
Ezután megadtam a megfelelő elérési utat a térképemhez, amit a saját könyvtáramban hoztam létre. Ez beállítja az autonóm navigációhoz szükséges összes csomópontot. Egy másik

terminálon futtattam az Rviz-t, de az előre beállított vizualizációval, a következő parancsokkal:

- source /opt/ros/galactic/setup.bash
- ros2 run rviz2 rviz2 -d $(ros2 pkg prefix nav2_bringup) /share/nav2_bringup/rviz/nav2_default_view.rviz

Elméletileg, ha a robot nagyjából tudja, hogy hol van és milyen irányba néz, akkor össze tudja illeszteni a lézer adatokat a térképadatokkal és az út hátralévő részében igazodik A ROS 2 navigáció meg tudja határozni a robot pozícióját, de a kezdeti pozícióhoz egy kis segítségre van szüksége, ezt a segítséget a „2D Pose Estimate” gombra kattintva és egy nyíl rajzolásával biztosítottam. Először megjelöltem a robot pozícióját a térképen, majd felfelé húzva mutattam, a felfelé mutató nyíllal, azt az irányt jelezve, amerre a robotom néz. Ezáltal meglett az autonóm navigációhoz szükséges összes átalakításom.

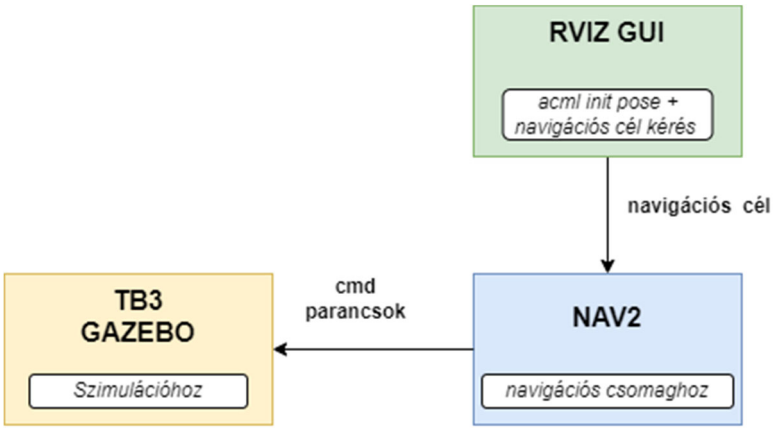
A mozgórobotom ezután megtervezett egy útvonalat és végighaladt azon, elkerülve az akadályokat. Mozgása során a robot folyamatosan értékelte aktuális helyzetét az eredeti tervhez képest és szükség szerint módosította. A Nav2 csomag útvonalat tervezett (helyi + globális) és elkezdte mozgatni a robotot, tehát a navigációt sikeresen végbement, amelyet a 22. ábra szemléltet.



22. ábra. A mozgórobot navigációja

4.1. Járőr robot megtervezése

Kezdésképpen, részletezem, mivel foglalkoztam az előző fejezetben. A rendszerem három nagy komponensből áll. Az első a mozgórobot szimulációja a Gazebo világban, a második a navigációs csomagom, a Nav2 és a harmadik pedig az RVIZ GUI volt, ezt a felépítést a 23.ábra jeleníti meg. Az előző projektben a mozgórobot szimulációjával kezdtem, ezután futtatam a Nav2-t, aminek az volt a feladata, hogy navigációs utasításokat adjon a robotnak.

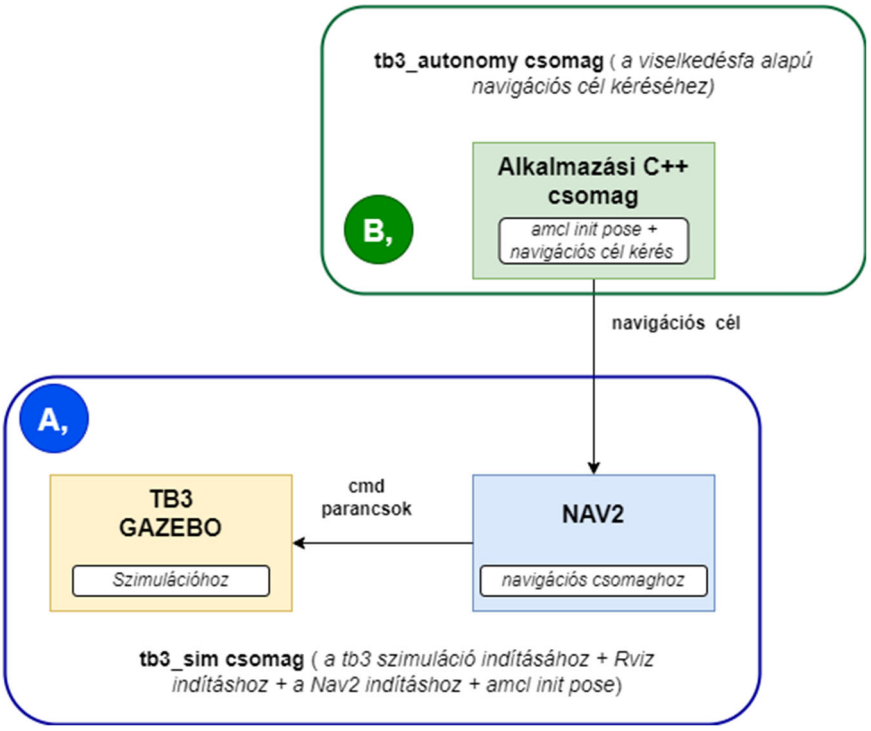


23. ábra: Az előző feladat felépítése stukturálisan

A harmadi komponensnél pedig elindítottam az RVIZ GUI-t, ahol két dologgal foglalkoztam. Kezdésként az *amcl (Adaptive Monte Carlo Localization)* parancsot használtam kezdeti pozíció megadására, ami a Nav2 részéhez tartozott, második lépésként viszont navigációs kódot adtam a robotnak, hogy tudjon a már definiált pozíciókba mozogni.

A feladatom második részében használtam az előző rendszerem felépítését, szintén tartalmazott egy mozgórobot szimulációt Gazebo-ban illetve a második komponenst is, nevezetesen a Nav2-t. A harmadik szegmenset, az RViz GUI-t lecseréltem a saját alkalmazási kódomra és az új parancsot alkalmaztam. Az alkalmazási kódban egy új csomagot hoztam létre és a Nav2-höz intéztem a robot mozgatását. Ezért viselkedés fákat alkalmaztam, mivel egy egyszerű viselkedési fa (Behavior Tree) létrehozásával minden levélcsomópont egy pozícióval egyezik meg a roboton. Így bemutatom, azt hogy hogyan hoztam létre viselkedéseket C++ programozási nyelv segítségével és hogyan használtam azokat a Nav2-vel való interakcióhoz. Fontos megjegyeznem, hogy ez a példa nem változtatta meg a Nav2 csomag belső viselkedésfáját. Mivel egy külső viselkedést alkottam meg, ami folyamatosan

kommunikál a Nav2 csomaggal, vagyis is újra és újra meghatározott pozíciókra fogja irányítja a robotot a *navigate to pose* parancs segítségével. Az új rendszer felépítésemet a 24. ábra prezentálja.



24. ábra: Az új megváltoztatott félépítés az alkalmazási kóddal

Azonban ahelyett, hogy a feladat elején használt indítási fájlt alkalmaztam volna a szimulációhoz és a robot elindításához is, létrehoztam egy saját indítási fájlt. Tehát egy új indítási fájlt generáltam az első komponens számára, ami a Gazebo szimulációs beállításait is tartalmazza, valamint saját indítási fájlt kapott a második komponens is, a Nav2. Az előző feladatban egy terminálban indítottam el a navigációt, egy másikban az RVIZ-t, majd az RVIZ-ből használtam az *amcl*-t a kezdeti pozíció beállításához. Ezért akartam azt, hogy mindezt egyetlen indítási fájlban legyen és csak két terminált használjak az összes beállításhoz, ezért így létrehoztam egy saját indítási fájlt a Gazebo szimulációhoz, a Nav2-höz és az RVIZ beállításához is. Az infrastruktúra nevű csomagot alkottam meg, ami magába foglalja ezeket a beállításokat. Ezt a csomagot "tb3 sim"-nek neveztem el. Az első csomag a szimulációs és a Nav2 beállításokból állt.

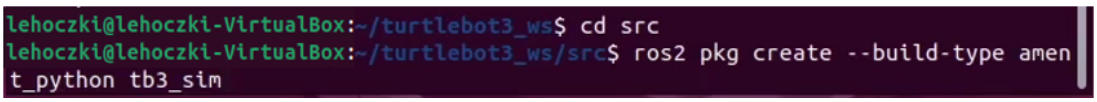
Az első részben az 24. ábrán „A”-val jelölt halmazt alkottam meg. Tehát a feladatomat egy terminál megnyitásával kezdtem és létrehoztam a *mkdir* parancs segítségével a „turtlebot3_ws” könyvtáramat. Összeszedtem az összes szükséges mozgórobothoz tartozó

Szent István Campus, Gödöllő
Cím: 2100 Gödöllő, Páter Károly utca 1.
Tel.: +36-28/522-000
Honlap: https://godollo.uni-mate.hu

csomagot. Mivel a ROS2 Humble verziót használtam, ezért a *turtlebot3.repos*-t, a vcs scripts fájlt importáltam be. Beletettem az összes szükséges kódot és az összes csomagot beolvastattam így ezáltal újraépítem a terminálomat, ennek eredményeképpen a teljes turtlebot3 csomag létrejött, amely a szimuláció fútatásához szükséges. Miután befejeztem az előkészületeket, készítettem egy új csomagot a munkaterületemen. A következő parancs alkalmazásának segítségével, a „*colcon build –symlink-install*”-al, sikerült létrehoznom mind a 16 csomagot.

- *A tb3_sim csomag létrehozása:*

Következő lépésként megalkottam az új csomagot, amely a szimulációs beállítások elindításért és a Nav2 konfigurálásáért felelt, az új csomagot *tb3_sim*-nek neveztem el, ez egy python csomag, mivel egy python szkriptem futtattam az *amcl* kezdeti pozíció beállításához, az új csomag készítését 25. ábra mutatja, valamint a Visual Studioban felépített tb3_sim csomag könyvtárainak illetve fájljainak tartalmát mellékletben hivatkoztam meg.



25. ábra: A tb3_sim csomag létrehozása

Az új, egyedi csomag készítése után a Visual Studio-t indítottam el, mivel itt hoztam létre egy új indítófájlt, hogy aktíváljam a Gazebo szimulációt a robot számára, a fájlt *"turtlebot3_world.launch.py*" neveztem el, ez egy egyszerű módszer a robotállapot kiadásához. Az új indító fájlom magába foglalja összes olyan importálást, amire szüksége van az operációs rendszernek, hogy megkapja az összes fájl elérési útját. Továbbá tartalmazza a „*generate_launch_description*” funkciót, a "turtlebot3_gazebo" csomag "launch" fájlkönyvtárára, hogy eltudjam indítani később a "robot state publisher"-t. A "gazebo_ros" csomagot is használtam a Gazebo pálya útvonalának megszerzéséhez. Amit még lefedett az új indítás fájlom az konfigurációs értékek, alapvetően ezek olyan pozíció értékek, ahol azt akarom, hogy a robot olyan szimulációs időt használjon amelyre igaz az érték. Végezetül a fájl tartalmaz teljes mozgórobot létrehozására szolgáló parancsot, kezdeti pozíció értékekkel. Az indítófájl egyszerűen csak elindította a "turtlebot3" pályát, amit a Gazebo rendelkezésre bocsátott. Az indító fájl felépítését python kódok generálásával alkottam meg, így a tb3_sim csomag setup.py fájlban is végre hajtatottam a módsítást, nevezetesen az összes indítófájlnak

Szent István Campus, Gödöllő
Cím: 2100 Gödöllő, Páter Károly utca 1.
Tel.: +36-28/522-000
Honlap: https://godollo.uni-mate.hu

át kellet kerülnie a megosztott könyvtárba, így megadtam az elérési útját, ami akkor lesz használatban, amikor ténylegesen futtattom rendszeremet.

Egy új csomópontot hoztam létre amely azért felelős, hogy definiálja a robot kezdeti pozícióját az *amcl*-nek. A csomópotom egy új fájl a tb3_sim csomagban és *set_init_amcl_pose.py* nevet adtam neki. Inicializáltam egy kiadót, az *amcl* pozíció információk elküldéséhez. A csomópont az *intital_pose_topic* segítségével, megkapta a kezdeti pozíció információkat majd utána kinyerte a szükséges pozíció értékeket (x, y, theta, cov). Négy paraméterem van az X, Y Theta és kovariancia, Az összes szükséges paramétert hozzáadtam a paraméter blokkba, ahol az alapértelmezett értékeket használtam. Az "*initial_pose*" paramétereket is hozzáadtam a paraméter blokkhoz, mivel a robot kezdeti pozícióját is be kell állítanom.

A másik új indítófájl létrehozásával folytattam a feladatot, az új indítófájl a Nav2 beállításokhoz, amit a "nav2.launch.py" neveztem el. Ehhez szintén használtam a "generate_launch_description" függvényt és importáltam a "node" csomagot, mert egy saját node-ot alkalmaztam, hogy publikáljam az "initial_pose" témát, amelyet a Nav2 külsőleg is elfogad. Az indítófájl tartalmazza a térbeli transzformációhoz szükséges csomagokat és létrehoz egy node-ot az "initial_pose" témának. A node segítségével küldhettem el a robot kezdeti pozícióját a Nav2 számára és ezáltal beállítottam a robot kezdeti pozícióját. Az indítófájl tartalmazza az "initial_pose" témát és annak paramétereit, amelyek lehetővé teszik a kezdeti pozíció beállítását. Most már készen áll arra a projektem, hogy futtassam a szimulációs beállításokat és a Nav2-t egyetlen indítási fájlból.

Újra elindítom a terminált és a a „*colcon build*” parancs futtatásának segítségével és sikerült létrehoznom mind a 17 csomagot. Ezután a szimulációs beállításokat aktiváltam, beexportáltam a mozgórobot modelljének elérési útját, a tb3_sim csomag tartalmát kilistáztattam. A csomag listában szerepel az általam létrehozott két indításifájl, a nav2.launch.py illetve a turtlebot3_world.launch.py is, ezeket a műveleti végrehajtásokat 26. ábra ábrázolja.

```
Summary: 17 packages finished [5.65s]
  3 packages had stderr output: tb3_sim turtlebot3_example turtlebot3_teleop
lehoczki@lehoczki-VirtualBox:~/turtlebot3_ws$ source ./install/setup.bash
lehoczki@lehoczki-VirtualBox:~/turtlebot3_ws$ export GAZEBO_MODEL_PATH=$GAZEBO_MODE
L_PATH:~/turtlebot3_ws/src/turtlebot3/turtlebot3_simulations/turtlebot3_gazebo/m
odels
lehoczki@lehoczki-VirtualBox:~/turtlebot3_ws$ export TURTLEBOT3_MODEL=waffle_pi
lehoczki@lehoczki-VirtualBox:~/turtlebot3_ws$ ros2 launch tb3_sim
-a                              -p
-d                              package.xml
--debug                         --print
--launch-prefix                 --print-description
--launch-prefix-filter          -s
map.yaml                        --show-all-subprocesses-output
-n                              --show-args
nav2.launch.py                  --show-arguments
--noninteractive                turtlebot3_world.launch.py
```

26. ábra: A robot modell exportálása illetve az indítófájlok kilistázása

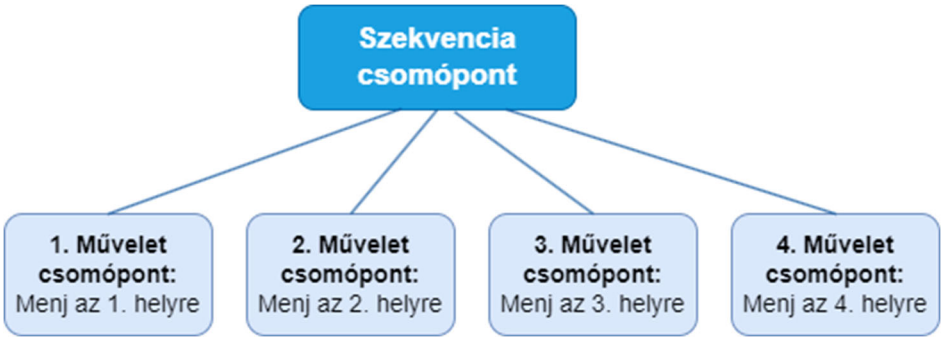
Ezután a két indítási fájlt indítottam el, amivel megjelent a mozgórobot szimulációja Gazebo ablakban illetve a Nav2 infrastruktúra beállításhoz szükséges Rviz felülete.
Így mindkét komponens egyetlen egy terminálba indul el. Ezzel befejeztem az első csomag létrehozását, amely a Gazebo szimulációs és a Nav2 beállításokat tartalmazza. A következő fejezetben a második csomagot fogom létrehozni, amely az alkalmazási kódot és a Behavior Tree-t foglalja magában.

- *A tb3_autonomy csomag előállítása:*

A projekt második részében, elkészítettem 24. ábrán mutatott „ B” részt, vagyis az alkalmazás csomagomat C++ programozási nyelven. Ezt a csomagot *tb3_autonomy* néven neveztem el, a csomag felelős a Nav2 célok küldésért a futó Nav2-nek, a második komponensnek. A Nav2 tovább adja a sebességérték parancs használatával a TB3 Gazebo szimulációs környezetnek, a robot mozgatásához, ezáltal a járőr robotomat oda irányítom ahová csak akarom, a megadatott pontok segítségével.

Az autonómia csomag magában foglalja, a *ROS 2 autonomy node*-t, a viselkedésfa vezérlő csomópontot, ahogyan már említettem viselkedésfát használtam a robot vezérlésére. Először is, megemlítem a ROS 2 autonómia csomópont, A ROS2 vagy ROS Tool-ban a csomagok olyan csomagok, amelyek általában az alapfunkciók felett vannak, mivel a ROS a közbenső réteg. Az ötletem az volt, hogy lehetővé tegye a folyamatok közötti kommunikációt. Ebben az esetben a C++ alkalmazás csomag egy folyamat és ehhez a kommunikációhoz üzeneteket, szolgáltatásokat, műveleteket stb. használtam. Így a szekvenciámban, egy szekvenciális csomópont és négy műveleti csomópont volt. Az alapvető funkciók a viselkedés fában vannak.

A viselkedésfám egy szekvencia csomópontból állt, amelyhez négy műveletcsomópont tartozott, ezt a felépítést a 27. ábra szemlélteti. Minden műveletcsomópont felelős azért, hogy a robotot egy adott helyre irányítsa. A feladatban a robotnak négy kijelölt helye van, ezt a négy pontot egy konfigurációs fájl segítségével határoztam meg.



27. ábra: A viselkedés fa felépítése

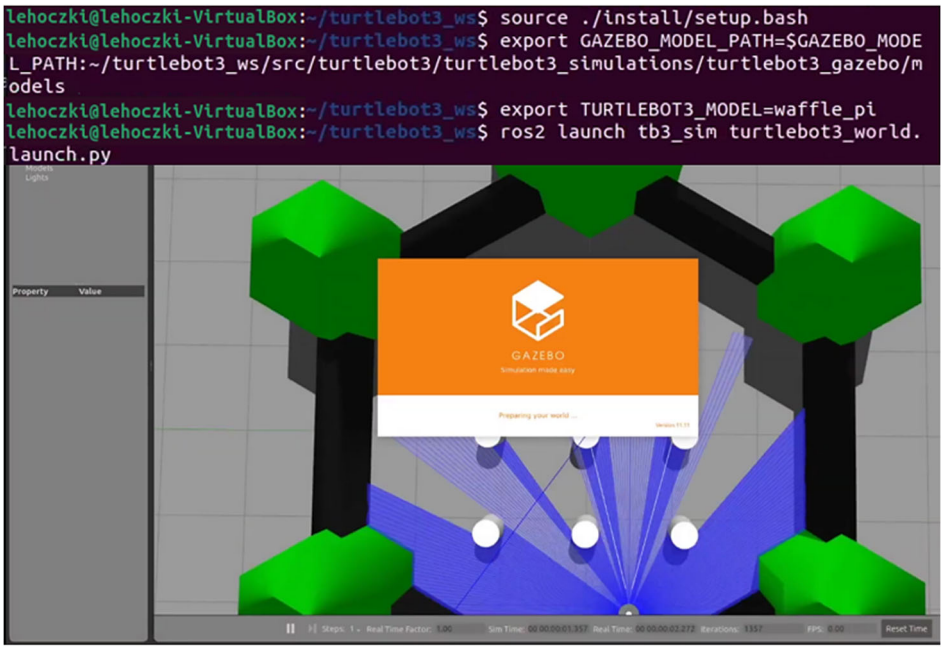
A viselkedés fát a behaviority.cpp könyvtár használatával alkottam meg. Tehát volt négy helyem, négy csomópontom, de mindegyik ugyanazt a működést használta, ezért létrehoztam egy osztályt és újra felhasználtam a négy csomóponthoz, mindegyik *node* a robot helyét vette be bementként, a bementei port alkalmazásával megkérdezte a robottól, hogy másik helyre mozogjon a bemeneti portól kapott adatok információ alapján.

Következő lépésként a *tb3_autonomy* csomagot hoztam létre a „*ros2 pkg create –build -type ament_cmake tb3_autonomy -dependecies rclcpp*” paranccsal. A csomagot a következő fájlokat tartalmazza: az *include dir*, az *src dir* (itt található az alapvető funkciók, indítófájlok), a *bt_xml dir* (ahol létre kellet hoznom egy XML-fájt a viselkedéshez, a viselkedésfát), a *config dir* (itt megalkottam egy konfigurációs *yaml* fájt, amelyben a négy hely van, ahová a robotnak mennie kell) , a *CmakeLists. txt* és végül *package.xml,* amiket a Visual Studio Code kódszerkesztőben írok meg. Tehát a terminálban a *„code .”*parancs futtatásával elindítottam a Visual Studio-t. A Visual Studioban elkészített *tb3_autonomy* csomag tartalmait, vagyis előbb említett könyvárak és fájlok tartalmi kivonatait mellékletben tüntettem fel.

A „*colcon biuld --symlink install*” paranccsal sikerült mind a 18 csomagot futtatni. Tehát van egy mozgórobotunk egy világban, a Nav2 működik és fut, az alkalmazási kódunkból, amely tartalmazza az összes számítást egy valós projekthez. A robotot egy, kettő, három és négy helyre tudom mozgatni egy külső viselkedés fa segítségével, amit C++-ban hoztam létre. Összesen három terminálra van szükségem a projekt futtatásához, az első

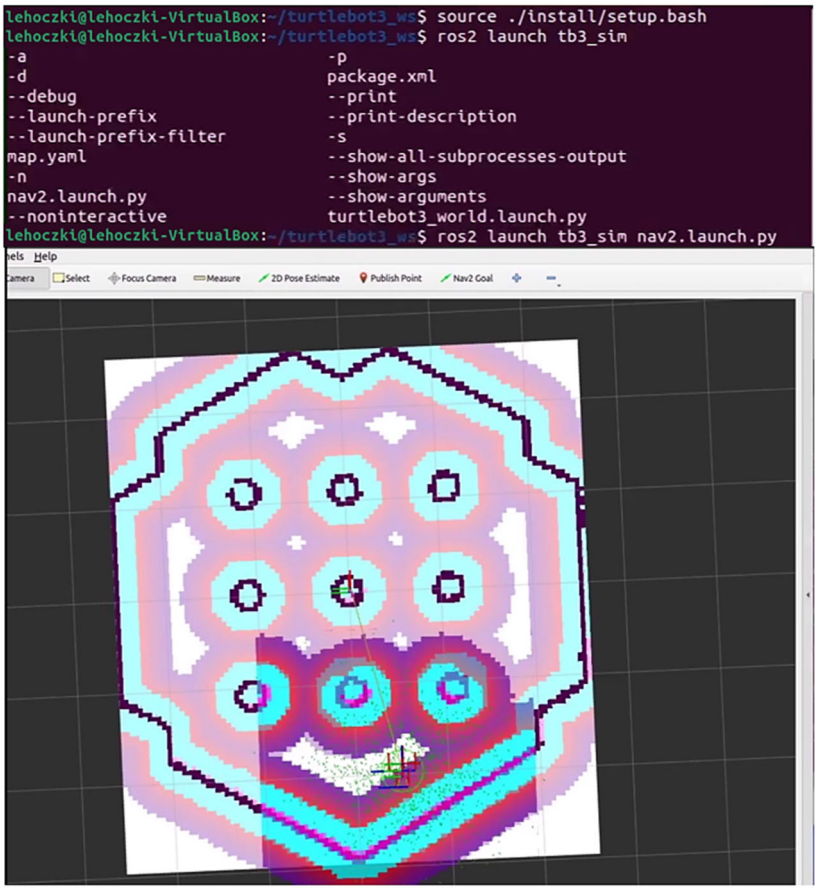
terminálban a robot modellt a Gazebo világban, a második terminálban a navigációs infrastruktúrát és végül a harmadik terminálban az autonómia csomópontot indítom el.

Megnyitottam az első terminált, ahol kezdésként a Gazebo szimuláció parancsot indítottam. A turtlebot3 modellt kiexportáltam, ehhez wallfe_pi -t alkalmaztam valamint az előző feladatrészben megalkotott turtlebot3_world.launch.py indítófájt futtattam, így készen állt a szimulációm. A parancsok indításait a munkafelületen valamint a Gazebo szimulációs ablak megnyitását a 28 .ábra jeleníti meg.



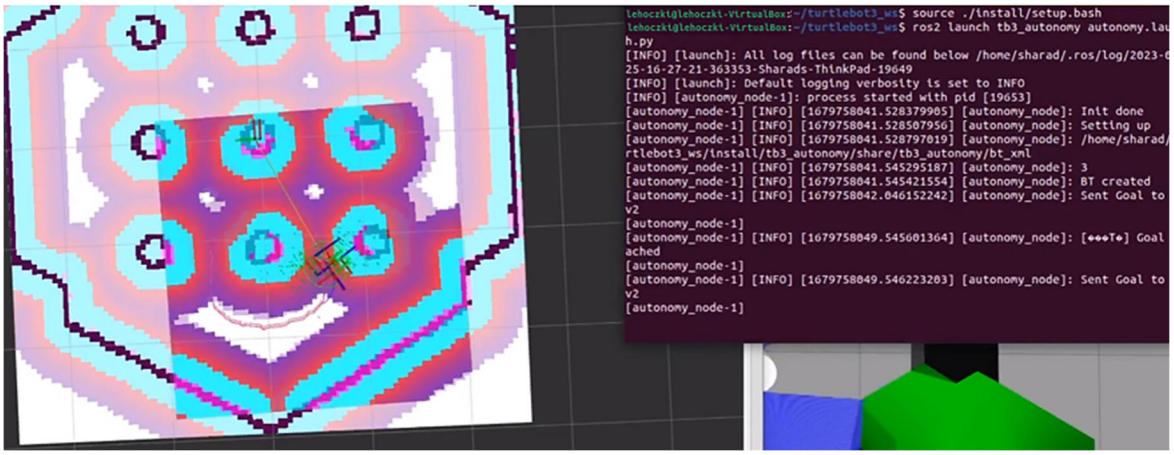
28. ábra: A robot modell a Gazebo szimulációba

A Nav2 infrastruktúrát már beállítottam a feladat első részében, ami szintén a tb3_sim csomag része volt. Ismét egy új terminált indítottam el a *tb3_sim nav2.launch.py* utasítást aktiváltam ezáltal a RViz vizualizáció program működésbe lépet, amit a *29*. ábra prezentál.



29. ábra: RViz felület elindítása

Egy másik terminálban elindítottam autonómia csomópontomat a csomagban található indítófájl segítségével, mármint futtatam *tb3_autonomy.launch.py* az indítófájl parancsot, így látható, hogy a robotom az első pontról a második pontra halad, a másodikról a harmadikra mozog és a harmadiktól a negyedik pontra megy és így tovább, tehát önjárást végez a mozgórobotom. A terminálban alkamazott parncsokat illetve az RViz szimulációs felületen látható jarőr robot mozgásának vonalát *30*. ábra mutatja.



30. ábra: Az RViz vizualizációban járőr robot valamint a terminálban futatott parancsok

Összefoglalva az autonóm navigáció beállítása című fejezetben, egy járő robot létrehozását mutattam be a ROS 2 Navigation keretrendszer segítségével. A Gazebo-ban szimulált robotot úgy terveztem, hogy ismételten végigjárjon egy négy pont által meghatározott területet. Az alap szimulációban három összetevőm volt, a TurtleBot 3 szimuláció, a Nav2 csomag és az Rviz GUI. A járőr robot feladatban bemutattot rendszerem három fő összetevőből állt, a Gazebo szimuláció TurtleBot-tal, a ROS 2 Navigation csomag (Nav2) és a harmadik komonens egy alkalmazási réteg volt, amely C++ nyelven készült viselkedésfát (Behavior Tree) használtával. A Gazebo szimuláció magában foglalt egy világot és egy TurtleBot3 mozgórobotot. A navigációs utasításokat a Nav2 csomag biztosította. Az általam létrehozott alkalmazási réteg viselkedésfát használt a robot mozgásának irányításához a meghatározott négy pont között. Két ROS2 csomagot hoztam létre ezen példa céljából, a tb3_sim a szimulációs felállásért és a tb3_autonomy az alkalmazási kódért felelt, amely viselkedésfát (Behavior Tree) használt.

Szent István Campus, Gödöllő
Cím: 2100 Gödöllő, Páter Károly utca 1.
Tel.: +36-28/522-000
Honlap: https://godollo.uni-mate.hu

5. Következtetések és javaslatok

A szakdolgozat témám fejlesztésének következő lépése lehet, akár a ROS2 és a MATLAB közötti interfész létrehozása, amelyhez különböző tanulmányok már készültek és amelyeket a 2. fejezetben említettem. A MATLAB környezet integrálása előnyös a ROS2-ből származó üzenetek és szenzoradatok vizualizálásához. A ROS2-ben különböző típusú algoritmusokat használnak, például tervezési és döntéshozatali algoritmusokat, érzékelési algoritmusokat, vezérlő algoritmusokat vagy érzékelési algoritmusokat egy autonóm rendszer számára. A MATLAB és a ROS2 közötti kommunikáció lehetséges az asztali prototípusok segítségével, amely lehetőséget biztosít fájlok kinyerésére és parancsok küldésére.

A kommunikáció másik formája az, hogy a tesztelésre kifejlesztett tervből automatikusan generálhatók C++ alapú futtatható ROS2 csomópontok. Ezek a generált csomópontok integrálhatók a ROS2 rendszerbe, így a célrendszeren futtathatók a MATLAB és a Simulink függőségek nélkül. A ROS toolbox segítségével megvalósítható a MATLAB/Simulink és a ROS2 közötti kommunikáció.

Szent István Campus, Gödöllő
Cím: 2100 Gödöllő, Páter Károly utca 1.
Tel.: +36-28/522-000
Honlap: https://godollo.uni-mate.hu

6. Összefoglalás

Gépipari automatizálási szakmérnöki képzésben kidolgozott szakdolgozat témám a mozgórobotok navigációs algoritmusának fejlesztése volt, amelynek célja, a mozgórobot sík felületen történő mozgása a kijelölt kezdőpontól, egészen a végpontig, különböző akadályokat elkerülve a megtett pálya során. A dolgozatomban egy járőr mozgórobot alkalmazását készítem el. Először az autonóm navigációt hoztam létre a mozgórobot számára majd az általam C++ megírt indítóprogramok és viselkedésfák segítségével a meghatározott 4 pontban a robot folyamatos önjárást végez. amelynek célja, a robotnak, hogy az A pontból el kell jutnia B pontba. A mozgórobot modellen keresztül tesztelhetem python kódjaimat Gazebo szimulációs környezetben. A szimulációhoz több környezetet fejlesztettem ki. A szakdolgozatomba használt mozgórobot, nevezetesen Turtlebot3 volt, a robot rendelkezik az összes fizikai tulajdonságokkal, megfelelő érzékelőkkel, kamerákkal valamint kódolókkal szerelték fel. A szimulációs környezet létrehozásához használt Gazebo szoftver hatékony interfészt tartott fenn a renderelési képességekhez. A szimuláció elvégzéséhez figyelembe vettem különböző könyvtárakat és szkripteket. Ez a szimulációs rendszer más robotmodellekhez, akár valós modellekhez is könnyen alkalmazható az üzenetek és szolgáltatások módosításával. A Gazebóval integrált ROS2 hatékonyabb és vonzóbb munkavégzést tett lehetővé. A ROS2 sokoldalú és hatékony eszköz a robotikai kutatásban, valamint olyan alkalmazásokban, mint a mobil robotok navigációinak kiépítésében. A ROS2 vizualizációs eszközei, mint az RViz és a SLAM segítségével vizualizáltam a mozgórobot különböző tulajdonságait, például az odometriát. A ROS2 eszköztár jó választásnak bizonyult mobil robotok autonóm programjainak megvalósításához, elérhetővé téve azt még azoknak is, akik nem ismerik a Linuxot vagy a ROS2-őt.

Szakdolgozat elkészítése alatt különböző operációs rendszerek, szimulátorok és könyvtárak beépítését, Python illetve C++ forráskódú parancsok alkalmazását, egy fedél alá hozását hajtottam végre, ezáltal rengeteg új ismerettel gazdagodtam.

Szent István Campus, Gödöllő
Cím: 2100 Gödöllő, Páter Károly utca 1.
Tel.: +36-28/522-000
Honlap: https://godollo.uni-mate.hu

7. Summary

My thesis topic was the development of a navigation algorithm for moving robots, which aims to move the robot on a flat surface from the starting point to the end point, avoiding various obstacles along the way. In my thesis, I prepare an application of a patrol moving robot. I first created autonomous navigation for the moving robot and then using the startup programs and behavior trees I wrote in C++, the robot performs continuous self-pacing at the 4 points defined, with the goal of the robot to get from point A to point B. I can test my python code in Gazebo simulation environment using the moving robot model. I developed several environments for the simulation. The moving robot used in my thesis was namely Turtlebot3, the robot has all the physical features, equipped with appropriate sensors, cameras as well as encoders. The Gazebo software used to create the simulation environment maintained an efficient interface for rendering capabilities. I considered different libraries and scripts to perform the simulation. This simulation system can be easily adapted to other robot models, even real models, by modifying messages and services. ROS2, integrated with Gazebo, has allowed me to work more efficiently and attractively. ROS2 is a versatile and powerful tool in robotics research, as well as in applications such as building navigation systems for mobile robots. I used ROS2 visualization tools such as RViz and SLAM to visualize different properties of the mobile robot, such as odometry. The ROS2 toolkit has proven to be a good choice for implementing autonomous programs for mobile robots, making it accessible even to those who are not familiar with Linux or ROS2.

During my thesis, I have implemented different operating systems, simulators and libraries, used Python and C++ source code commands, and brought them under one roof, thus gaining a lot of new knowledge.

NYILATKOZAT

a záródolgozat/szakdolgozat/diplomadolgozat/portfólió[1] nyilvános hozzáféréséről és eredetiségéről

A hallgató neve: Lehoczki Viktor

A Hallgató Neptun kódja: DEJZQJ

A dolgozat címe: Mozgó robot navigációs algoritmusának fejlesztése ROS 2 környezetben

A megjelenés éve: 2023

A tanszék neve: Mechatronika Tanszék

Kijelentem, hogy az általam benyújtott záródolgozat/szakdolgozat/diplomadolgozat/portfólió[2] egyéni, eredeti jellegű, saját szellemi alkotásom. Azon részeket, melyeket más szerzők munkájából vettem át, egyértelműen megjelöltem, s az irodalomjegyzékben szerepeltettem.

Ha a fenti nyilatkozattal valótlant állítottam, tudomásul veszem, hogy a Záróvizsga-bizottság a záróvizsgából kizár és a záróvizsgát csak új dolgozat készítése után tehetek.

A leadott dolgozat, mely PDF dokumentum, szerkesztését nem, megtekintését és nyomtatását engedélyezem.

Tudomásul veszem, hogy az általam készített dolgozatra, mint szellemi alkotás felhasználására, hasznosítására a Magyar Agrár- és Élettudományi Egyetem mindenkori szellemitulajdon-kezelési szabályzatában megfogalmazottak érvényesek.

Tudomásul veszem, hogy dolgozatom elektronikus változata feltöltésre kerül a Magyar Agrár- és Élettudományi Egyetem könyvtári repozitori rendszerébe.

Kelt: 2023 év 11 hó 10 nap

Lehoczki Viktor
Hallgató aláírása

[1] A megfelelő dolgozattípus meghagyása mellett a többi típus törlendő.
[2] A megfelelő dolgozattípus meghagyása mellett a többi típus törlendő.

MAGYAR AGRÁR- ÉS ÉLETTUDOMÁNYI EGYETEM

Szent István Campus, Gödöllő
Cím: 2100 Gödöllő, Páter Károly utca 1.
Tel.: +36-28/522-000
Honlap: https://godollo.uni-mate.hu

NYILATKOZAT

Alulírott *Lehoczki Viktor*, a Magyar Agrár- és Élettudományi *Egyetem, Szent István Egyetem Campus, Gépipari automatizálási szakmérnöki* szak levelező tagozat végzős hallgatója nyilatkozom, hogy a dolgozat saját munkám, melynek elkészítése során a felhasznált irodalmat korrekt módon, a jogi és etikai szabályok betartásával kezeltem. Hozzájárulok ahhoz, hogy Szakdolgozatom egyoldalas összefoglalója felkerüljön az Egyetem honlapjára és hogy a digitális verzióban (pdf formátumban) leadott dolgozatom elérhető legyen a témát vezető Tanszéken/Intézetben, illetve az Egyetem központi nyilvántartásában, a jogi és etikai szabályok teljes körű betartása mellett.

A dolgozat állam- vagy szolgálati titkot tartalmaz: igen nem*

Kelt: 2023 év 11 hó 10 nap

Lehoczki Viktor
Hallgató

NYILATKOZAT

A dolgozat készítőjének konzulense nyilatkozom arról, hogy a Záródolgozatot/Szakdolgozatot/Diplomadolgozatot áttekintettem, a hallgatót az irodalmi források korrekt kezelésének követelményeiről, jogi és etikai szabályairól tájékoztattam.

A Záródolgozatot/Szakdolgozatot/Diplomadolgozatot záróvizsgán történő védésre javaslom / nem javaslom*.

A dolgozat állam- vagy szolgálati titkot tartalmaz: igen nem*

Kelt: 2023 év 11 hó 10 nap

Belső konzulens

***Kérjük a megfelelőt aláhúzni!**

KONZULTÁCIÓS NYILATKOZAT

Lehoczki Viktor (hallgató, Neptun azonosítója:DEJZQJ) konzulenseként nyilatkozom arról, hogy a szakdolgozatot[1] áttekintettem, a hallgatót az irodalmi források korrekt kezelésének követelményeiről, jogi és etikai szabályairól tájékoztattam.

A záródolgozatot/szakdolgozatot/diplomadolgozatot/portfóliót a záróvizsgán történő védésre javaslom / nem javaslom[2].

A dolgozat állam- vagy szolgálati titkot tartalmaz: igen nem*[3]

Kelt: 2023 év 11 hó 10 nap

Tóth János
egyetemi tanársegéd
belső konzulens
MATE SZIC Műszaki Intézet
Mechatronika Tanszék

[1] A megfelelő dolgozattípus meghagyása mellett a többi típus törlendő.
[2] A megfelelő aláhúzandó.
[3] A megfelelő aláhúzandó.

9. Felhasznált irodalom jegyzéke

[1] A. K. Guruji, H. Agarwal, and D. K. Parsediya (2016): *Time-efficient A* Algorithm for Robot Path Planning*, Procedia Technology, vol. 23, pp. 144–149, Letöltés dátuma: 2023. 08.31. forrás: https://www.scirp.org/(S(lz5mqp453edsnp55rrgjct55.))/reference/referencespapers.aspx?referenceid=1911901

[2] A. Koubâa (2016): *Robot operating system (ROS): The complete reference. Volume 1* / Anis Koubâa, Editors. Cham: Springer. DOI: 10.1007/978-3-319-26054-9

[3] A. Elshamli, H. A. Abdullah, and S. Areibi,(2004): *Genetic algorithm for dynamic path planning*, in Canadian conference on electrical and computer engineering 2004 =Conférence Canadienne en génie électrique et informatique 2004, Niagara Falls, Ont., Canada, pp. 677–680. Letöltés dátuma: 2023. 09.10. forrás: https://www.researchgate.net/publication/4096948_Genetic_algorithm_for_dynamic_path_planning

[4] Automaticaddison.com honlap, Letöltés dátuma: 2023.09.12. forrás: ROS 2 Architecture Overview – Automatic Addison

[5] B. Frank, M. Becker, C. Stachniss, W. Burgard, and M. Teschner, (2008): *Efficient path planning for mobile robots in environments with deformable objects*, in Robotics and Automation, 2008, ICRA 2008, IEEE International Conference on: Date, 19-23 May, 2008, Pasadena, CA, USA, pp. 3737–3742. DOI:https://researchr.org/publication/icra%3A2008

[6] B. N, W. N. W.A.J, S. N, and M. Z, (2011): *A Simple Local Path Planning Algorithm for Autonomous Mobile Robots*, International Journal of System Applications, Engi-neering and Development, vol. 5, no. 2, pp. 1–12, https://www.universitypress.org.uk/journals/saed/19-671.pdf

[7] Carol Fairchild, Dr. Thomas L. Harman, (2016): *ROS robotics by example: Bring life to your robot using ROS robotic applications* /, Bir-mingham, UK: Packt Publishing,

[8] C. Hofner and G. Schmidt, (1994): *Path planning and guidance techniques for an autono-mous mobile cleaning robot, in IROS* '94: Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems : advanced robotic systems and the real world 1994, Federal Armed Forces University, Munich, Germany, Munich,

Germany, pp. 610–617. DOI: Letöltés dátuma: 2023.09.05. forrás: https://doi.org/10.30880/ijie.2021.13.02.020

[9] C. W. Warren, (1993): *Fast path planning using modified A* method,* in Robotics and automation: International conference : Papers, Atlanta, GA, USA, pp. 662–667. DOI: 10.1007/978-3-540-72383-7_83

[10] D. Ferguson and A. Stentz, (2006): *Anytime RRTs*, IEEERSJ International Con-ference on Intelligent Robots and Systems: Beijing, China, pp. 5369–5375. Letöltés dátuma: 2023. 09.10. forrás: https://www.ri.cmu.edu/pub_files/pub4/ferguson_david_2006_4/ferguson_david_2006_4.pdf

[11] D. S. Yershov and S. M. LaValle, (2012): *Simplicial Dijkstra and A ∗ Algorithms: From Graphs to Continuous Spaces*, Advanced Robotics, vol. 26, no. 17, pp. 2065–2085,

[12] ElectronicDesign honlapja, *Ros 2 explained: Overview and features*, Letöltés dátuma: 2023.08.24.,https://www.electronicdesign.com/markets/automation/article/21214053/electronic-design-ros-2-explained-overview-and-features

[13] F. Haro and M. Torres, (2006): A Comparison of Path Planning Algorithms for Omni-Directional Robots in Dynamic Environments, IEEE 3rd Latin American Robotics Symposium: Santiago, Chile, 26-27 October 2006, Santiago, Chile, 2006, pp. 18–25. DOI: 10.1109/LARS.2006.334319

[14] F. Islam, J. Nasir, U. Malik, Y. Ayaz, and O. Hasan, (2012): RRT∗-Smart: Rapid convergence implementation of RRT∗ towards optimal solution, in International Conference on Mechatronics and Automation (ICMA), Chengdu, China, Chengdu, China, 2012, pp. 1651–1656. DOI: 10.1109/ICMA.2012.6284384

[15] G.Campion, G.Bastin, and B.D.AndrCa-Novel, (1993): *Structural properties and classification of kinematic and dynamic models of wheeled mobile robots*, IEEE transactions on robotics and automation, vol.12, no.1, 1996, PP.47-62. DOI:10.1109/ROBOT.1993.292023

[16] G.Indiveri,(2009): *Swedish wheeled omnidirectional mobile robots: kinematics analysis and control*, IEEE transactions on robotics, vol.25, no.1, PP.164-171., DOI:10.1109/TRO.2008.2010360

[17] H. M. Choset, Principles of robot motion, (2005): T*heory, algorithms, and implementation*, Howie Choset, Cambridge, Mass., London: MIT Press

[18] H. Wang, Y. Yu, and Q. Yuan, (2011): *Application of Dijkstra algorithm in robot path-planning*, Second International Conference on Mechanic Automation and Control Engineering: July 15-17, 2011, Inner Mongolia, 1, pp. 1067–1069. DOI:10.1109/MACE.2011.5987118

[19] I. Aguinaga, D. Borro, and L. Matey, (2008): Parallel RRT-based path planning for selective disassembly planning, Int J Adv Manuf Technol, vol. 36, no. 11-12, pp. 1221–1233, DOI: 10.1007/s00170-007-0930-2

[20] I. Noreen, Khan Amna, and Z. Habib, (2016): *A Comparison of RRT, RRT*, and RRT*-Smart Path Planning Algorithms,* International Journal of Computer Science and Network Security, vol. 16, no. 10, pp. 20–27, 2016.

[21] J. Barraquand, B. Langlois, and J.-C. Latombe, (1992.): Numerical potential field techniques for robot path planning, IEEE Trans. Syst., Man, Cybern., vol. 22, no. 2, pp. 224–241, 1992. DOI:10.1109/21.148426

[22] J. J. J. Kuffner, S. Kagami, K. Nishiwaki, M. Inaba, and H. Inoue, (2002): Dynamically-stable Motion Planning for Humanoid Robots, Autonomous Robots, vol. 12, no. 1, pp. 105–118, DOI:10.1023/A:1013219111657

[23] J.L. Guzman, M. Berenguel, F. Rodriguez and S. Dormido, (2008): *An interactive tool for mobile robot motion planning*, Robotics and autonomous systems, vol. 56, no.5, PP.396–409.

[24] J. van den Berg, D. Ferguson, and J. Kuffner, (2006): *Anytime path planning and replanning in dynamic environments*, IEEE International Conference on, Orlando, FL, USA, 2006, pp. 2366–2371. DOI: 10.1109/ROBOT.2006.1642056

[25] K. H. Sedighi, K. Ashenayi, T. W. Manikas, R. L. Wainwright, and H.-M. Tai, (2004): *Autonomous local path planning for a mobile robot using a genetic algorithm*, Port-land, OR, USA, 2004, pp. 1338–1345. DOI:10.3233/HIS-130184

[26] K. Takaya, T. Asa, V. Kroumov ja F. Smarandache, (2016): *Simulation environment for mobile robots testing using ROS and Gazebo*, 20th International Conference on System Theory, Control and Computing (ICSTCC), Sinaia, Romania, DOI: 10.1109/ICSTCC.2016.7790647

[27] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, (1996): *Probabilistic roadmaps for path planning in high-dimensional configuration spaces*, IEEE Trans. Robot. Automat., vol. 12, no. 4, pp. 566–580, DOI: 10.1109/70.508439

[28] L. Jaillet, J. Cortés, and T. Siméon, (2010): *Sampling-Based Path Planning on Configuration-Space Costmaps*, IEEE Trans. Robot., vol. 26, no. 4, pp. 635–646, DOI: 10.1109/TRO.2010.2049527

[29] L. Joseph (2015): *Mastering ROS for Robotics Programming. Birmingham*, Packt Publishing, Limited

[30] Mathworks honlapja, *Probabilistic Roadmaps (PRM*). Letöltés dátuma: 2023.09.08.https://www.mathworks.com/help/robotics/ug/probabilistic-roadmaps-prm.html

[31] M. Karova et al. (2015): *Path Planning Algorithm for Mobile Robot*, Proceedings of the 15th international conference on applied computer science, Letöltés dátuma: 2023. 08.28. forrás: https://www.semanticscholar.org/paper/Path-Planning-Algorithm-for-Mobile-Robot-Karova-Zhelyazkov/5942ea80bed12af7040efad6ccf950b4d2cc1001

[32] M. I. S. Julius Fusic ja Dr.K.Hariharan, (2018): *Path planning for car like mobile robot using robot operating system*, National Power Engineering Conference (NPEC), Madurai, India, DOI:10.1109/NPEC.2018.8476806

[33] M. Quigley et al. (2009): *ROS: an open-source Robot Operating System*, Letöltés dátuma: 2023. 09.04. forrás: https://www.researchgate.net/publication/233881999_ROS_an_open-source_Robot_Operating_System

[34] M. Quigley, B. Gerkey, and W. D. Smart, (2015): Programming robots with ROS: A practical introduction to the robot operating system, Morgan Quigley, Brian Gerkey, William D. Smart. Beijing

[35] N. Achour and M. Chaalal, (2008): *Mobile Robots Path Planning using Genetic Algorithms*, in Robotics and Automation, IEEE International Conference pp. 111–145. Letöltés dátuma: 2023. 09.10. forrás: https://www.researchgate.net/publication/268006310_Mobile_Robots_Path_Planning_using_Genetic_Algorithms

[36] P. Vyavahare, S. Jayaprakash ja K. Bharatia, (2019): *Construction of URDF model based on open source robot dog using Gazebo and ROS*, Dubai, United Arab Emirates: IEEE, 2019, p. 5. DOI:10.1109/ICASET.2019.8714265

[37] R.Dhaouadi and A.A.Hatab, (2013): *Dynamic modelling of differential-drive mobile robots using lagrange and newton-euler methodologies: a unified framework*, Advances in robotics and automation, vol.2, no.2, PP.2-7. DOI: 10.4172/2168-9695.1000107

[38] R. Bohlin and L. E. Kavraki, (2000) *Path planning using lazy PRM*, IEEE inter-national conference on robotics and automation, San Francisco, 521-528 vol.1. Letöltés dátuma: 2023. 09.04. forrás: https://www.researchgate.net/publication/233881999_ROS_an_open source_Robot_Operating_System

[39] Reviewmylife.co.uk honlap, Dijkstra's Algorithm in C++, Letöltés dátuma: 2023. 09.06. http://www.reviewmylife.co.uk/blog/2008/07/15/dijkstras-algorithm-code-in-c/.

[40] ROS, ROS.org honlap, *Powering the World's Robots*. forrás: http://www.ros.org/.

[41] S.A.Stoeter and N.Papanikolopoulos, (2006): *Kinematic motion model for jumping scout robots*, IEEE transactions on robotics, vol. 22, no. 2, PP. 398-403. DOI: 10.1109/TRO.2006.862483

[42] S. Karaman and E. Frazzoli,(2011): *Sampling-based algorithms for optimal motion planning*, vol. 30, no. 7, pp. 846–894, forrás: https://arxiv.org/abs/1105.1186

[43] S. M. Ayazi, M. F. Mashhorroudi, and M. Ghorbani, Eds., (2014): *Modifed A* Algorithm Implementation in the routing optimized for use in geospatial information system,* DOI:10.5194/isprsarchives-XL-2-W3-69-2014

[44] S. S and Dr. C. Chandrasekar, (2014.): *Modified Dijsktra's Shortest Path Algorit*hm, vol. 2, no. 11, pp. 6450–6456, Letöltés dátuma: 2023. 09.02 forrás: https://www.rroij.com/open-access/modified-dijkstras-shortest-path-algorithm.pdf,

[45] T. Arora, Y. Gigras, and V. Arora, (2014): *Robotic Path Planning using Genetic Algorithm in Dynamic Environment*, IJCA, vol. 89, no. 11, pp. 8–12, DOI:10.5120/15674-4422

[46] T. B. Lauwers, G. A. Kantor, and R. L. Hollis, (2006): *A dynamically stable single-wheeled mobile robot with inverse mouse-ball drive*, Proceedings of IEEE International conference on robotics and automation, Orlando, FL, DOI:10.1109/ROBOT.2006.1642139

[47] W.-G. Han, S.-M. Baek, and T.-Y. Kuc, (1997): *Genetic algorithm based path planning and dynamic obstacle avoidance of mobile robots*," Orlando, FL, USA, pp. 2747–2751. DOI:10.3390/s21237898

[48] W. Wu, Z. Q. Sen, J. B. Mbede, and H. Xinhan, *Research on path planning for mobile robot among dynamic obstacles*, inVancouver, Canada, pp. 763–767. DOI:10.1109/NAFIPS.2001.944699

[49] X. Wu and S. Ma, (2010*): CPG-based control of serpentine locomotion of a snake-like*

robot, Mechatronics, vol. 20, no. 2, PP. 326–334. DOI:10.1201/b11365-3

[50] Y. X. Choi H. and H. Kim. (2021): PiCAS: *New design of priority-driven chain-aware scheduling for ROS2*, in IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), DOI:10.1109/RTAS52030.2021.00028

[51] Z. Riaz, A. Pervez, M. Ahmer ja J. Iqbal, (2010): *A Fully Autonomous Indoor Mobile Robot using SLAM*, pp. 1-6, DOI:10.1109/ICIET.2010.5625691

[52] Z.Y. Bayraktaroglu, (2009): *Snake-like locomotion: Experimentations with a biologically inspired wheel-less snake robot*, Mechanism and machine theory, vol. 44, no.3, PP.591–602.

Szent István Campus, Gödöllő
Cím: 2100 Gödöllő, Páter Károly utca 1.
Tel.: +36-28/522-000
Honlap: https://godollo.uni-mate.hu

10. Mellékletek

Folder config

config\sim_house_locations.yaml

```yaml
# Defines object locations as [x, y, theta]
location1: [-1.0, -0.5, -2.356 ]
location2: [-1.0, 2, 0.785]
location3: [0.0, 0.5, 1.571]
location4: [0.0, -2.0, -1.571]

```

Folder launch

launch\nav2.launch.py

```python
#!/usr/bin/env python3
#
# Copyright 2019 ROBOTIS CO., LTD.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
# Authors: Joep Tool

import os
from time import sleep
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import IncludeLaunchDescription, ExecuteProcess
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch.substitutions import LaunchConfiguration

from launch_ros.actions import Node


def generate_launch_description():
  pkg_nav2_dir = get_package_share_directory('nav2_bringup')
  pkg_tb3_sim = get_package_share_directory('tb3_sim')

  use_sim_time = LaunchConfiguration('use_sim_time', default='True')
  autostart = LaunchConfiguration('autostart', default='True')

  nav2_launch_cmd = IncludeLaunchDescription(
      PythonLaunchDescriptionSource(
          os.path.join(pkg_nav2_dir, 'launch', 'bringup_launch.py')
      ),
      launch_arguments={
          'use_sim_time': use_sim_time,
          'autostart': autostart,
          'map': os.path.join(pkg_tb3_sim, 'maps', 'map.yaml')
      }.items()
  )

  rviz_launch_cmd = Node(
      package="rviz2",
      executable="rviz2",
      name="rviz2",
      arguments=[
          '-d' + os.path.join(
              get_package_share_directory('nav2_bringup'),
              'rviz',
              'nav2_default_view.rviz'
          )]
  )

  set_init_amcl_pose_cmd = Node(
      package="tb3_sim",
      executable="amcl_init_pose_publisher",
      name="amcl_init_pose_publisher",
      parameters=[{
          "x": -2.0,
          "y": -0.5,
      }]
  )

  ld = LaunchDescription()

  # Add the commands to the launch description
  ld.add_action(nav2_launch_cmd)
```

```
  ld.add_action(set_init_amcl_pose_cmd)
  ld.add_action(rviz_launch_cmd)

  return ld

```

launch\turtlebot3_world.launch.py

```
#!/usr/bin/env python3
#
# Copyright 2019 ROBOTIS CO., LTD.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
# Authors: Joep Tool

import os

from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import IncludeLaunchDescription
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch.substitutions import LaunchConfiguration


def generate_launch_description():
  launch_file_dir = os.path.join(
      get_package_share_directory('turtlebot3_gazebo'), 'launch')
  pkg_gazebo_ros = get_package_share_directory('gazebo_ros')
  pkg_tb3_sim = get_package_share_directory('tb3_sim')

  use_sim_time = LaunchConfiguration('use_sim_time', default='true')
  x_pose = LaunchConfiguration('x_pose', default='-2.0')
  y_pose = LaunchConfiguration('y_pose', default='-0.5')

  world = os.path.join(
      get_package_share_directory('turtlebot3_gazebo'),
      'worlds',
      'turtlebot3_world.world'
  )

  gzserver_cmd = IncludeLaunchDescription(
      PythonLaunchDescriptionSource(
          os.path.join(pkg_gazebo_ros, 'launch', 'gzserver.launch.py')
      ),
      launch_arguments={'world': world}.items()
  )

  gzclient_cmd = IncludeLaunchDescription(
      PythonLaunchDescriptionSource(
          os.path.join(pkg_gazebo_ros, 'launch', 'gzclient.launch.py')
      )
  )

  robot_state_publisher_cmd = IncludeLaunchDescription(
      PythonLaunchDescriptionSource(
          os.path.join(launch_file_dir, 'robot_state_publisher.launch.py')
      ),
      launch_arguments={'use_sim_time': use_sim_time}.items()
  )

  spawn_turtlebot_cmd = IncludeLaunchDescription(
      PythonLaunchDescriptionSource(
          os.path.join(launch_file_dir, 'spawn_turtlebot3.launch.py')
      ),
      launch_arguments={
          'x_pose': x_pose,
          'y_pose': y_pose
      }.items()
  )

  ld = LaunchDescription()
```

```
75
76    ld.add_action(gzserver_cmd)
77    ld.add_action(gzclient_cmd)
78    ld.add_action(robot_state_publisher_cmd)
79    ld.add_action(spawn_turtlebot_cmd)
80
81    return ld
82
```

Folder maps

maps\map.yaml

```yaml
image: map.pgm
mode: trinary
resolution: 0.05
origin: [-2.96, -2.6, 0]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.25

```

Folder tb3_sim

tb3_sim__init__.py

```

```

tb3_sim\set_init_amcl_pose.py

```python
import time
import rclpy
from rclpy.node import Node
import transforms3d
from geometry_msgs.msg import PoseWithCovarianceStamped


class InitAmclPosePublisher(Node):
  def __init__(self):
    super().__init__("init_amcl_pose_publisher")

    self.declare_parameter("x", value=0.0)
    self.declare_parameter("y", value=0.0)
    self.declare_parameter("theta", value=0.0)
    self.declare_parameter("cov", value=0.5**2)

    self.publisher = self.create_publisher(
        PoseWithCovarianceStamped,
        "/initialpose",
        10,
    )

    while(self.publisher.get_subscription_count() == 0):
      self.get_logger().info("Waiting for AMCL Initial Pose subscriber")
      time.sleep(1.0)

  def send_init_pose(self):
    x = self.get_parameter("x").value
    y = self.get_parameter("y").value
    theta = self.get_parameter("theta").value
    cov = self.get_parameter("cov").value

    msg = PoseWithCovarianceStamped()
    msg.header.frame_id = "map"
    msg.pose.pose.position.x = x
    msg.pose.pose.position.y = y
    quat = transforms3d.euler.euler2quat(0, 0, theta)
    msg.pose.pose.orientation.w = quat[0]
    msg.pose.pose.orientation.x = quat[1]
    msg.pose.pose.orientation.y = quat[2]
    msg.pose.pose.orientation.z = quat[3]

    msg.pose.covariance = [
        cov, 0.0, 0.0, 0.0, 0.0, 0.0,  # Pos X
        0.0, cov, 0.0, 0.0, 0.0, 0.0,  # Pos Y
        0.0, 0.0, 0.0, 0.0, 0.0, 0.0,  # Pos Z
        0.0, 0.0, 0.0, 0.0, 0.0, 0.0,  # Rot X
        0.0, 0.0, 0.0, 0.0, 0.0, 0.0,  # Rot Y
        0.0, 0.0, 0.0, 0.0, 0.0, cov   # Rot Z
    ]

    self.publisher.publish(msg)


def main(args=None):
  rclpy.init()
  initAmclPosePublisher = InitAmclPosePublisher()

  future = initAmclPosePublisher.send_init_pose()

  rclpy.spin(initAmclPosePublisher)

  initAmclPosePublisher.destroy_node()
  rclpy.shutdown()

```

setup.py

```python
from setuptools import setup
import os
from glob import glob

package_name = 'tb3_sim'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
        (os.path.join('share', package_name),
         glob('launch/*launch.[pxy][yma]*')),
        (os.path.join('share', package_name, 'config/'),
         glob('config/*')),
        (os.path.join('share', package_name, 'maps/'),
         glob('maps/*')),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='lehoczki',
    maintainer_email='lehoczki.viktor95@gmail.com',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'amcl_init_pose_publisher = tb3_sim.set_init_amcl_pose:main',
        ],
    },
)

```

Folder bt_xml

bt_xml\tree.xml

```
<!-- Behavior tree that sequentially navigates locations naively -->
<root main_tree_to_execute = "MainTree" >
  <BehaviorTree ID="MainTree">
    <Sequence name="sequence">
      <GoToPose name="go_to_location1" loc="location1" />
      <GoToPose name="go_to_location2" loc="location2" />
      <GoToPose name="go_to_location3" loc="location3" />
      <GoToPose name="go_to_location4" loc="location4" />
    </Sequence>
  </BehaviorTree>
</root>
```

Folder include

include\autonomy_node.h

```cpp
#include "rclcpp/rclcpp.hpp"
#include "behaviortree_cpp_v3/bt_factory.h"
#include "navigation_behaviors.h"
#include "ament_index_cpp/get_package_share_directory.hpp"

#include <tf2/LinearMath/Quaternion.h>

class AutonomyNode : public rclcpp::Node
{
public:
  explicit AutonomyNode(const std::string &node_name);
  void setup();
  void create_behavior_tree();
  void update_behavior_tree();

private:
  rclcpp::TimerBase::SharedPtr timer_;
  BT::Tree tree_;
};
```

include\navigation_behaviors.h

```cpp
#include "rclcpp/rclcpp.hpp"
#include "behaviortree_cpp_v3/behavior_tree.h"
#include "rclcpp_action/rclcpp_action.hpp"
#include "nav2_msgs/action/navigate_to_pose.hpp"

#include <tf2/LinearMath/Quaternion.h>
#include <tf2_geometry_msgs/tf2_geometry_msgs.h>

class GoToPose : public BT::StatefulActionNode
{
public:
  GoToPose(const std::string &name,
           const BT::NodeConfiguration &config,
           rclcpp::Node::SharedPtr node_ptr);

  using NavigateToPose = nav2_msgs::action::NavigateToPose;
  using GoalHandleNav = rclcpp_action::ClientGoalHandle<NavigateToPose>;

  rclcpp::Node::SharedPtr node_ptr_;
  rclcpp_action::Client<NavigateToPose>::SharedPtr action_client_ptr_;
  bool done_flag_;

  // Method overrides
  BT::NodeStatus onStart() override;
  BT::NodeStatus onRunning() override;
  void onHalted() override{};

  static BT::PortsList providedPorts();

  // Action Client callback
  void nav_to_pose_callback(const GoalHandleNav::WrappedResult &result);
};
```

Folder launch

launch\autonomy.launch.py

```python
import os

from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription

from launch_ros.actions import Node


def generate_launch_description():
  pkg_tb3_sim = get_package_share_directory('tb3_sim')
  pkg_tb3_autonomy = get_package_share_directory('tb3_autonomy')

  autonomy_node_cmd = Node(
      package="tb3_autonomy",
      executable="autonomy_node",
      name="autonomy_node",
      parameters=[{
          "location_file": os.path.join(pkg_tb3_sim, "config", "sim_house_locations.yaml")
      }]
  )

  ld = LaunchDescription()

  # Add the commands to the launch description
  ld.add_action(autonomy_node_cmd)

  return ld

```

Folder src

src\autonomy_node.cpp

```cpp
#include "autonomy_node.h"

using namespace std::chrono_literals;

const std::string bt_xml_dir =
    ament_index_cpp::get_package_share_directory("tb3_autonomy") + "/bt_xml";

AutonomyNode::AutonomyNode(const std::string &nodeName) : Node(nodeName)
{
  this->declare_parameter("location_file","none");

  RCLCPP_INFO(get_logger(), "Init done");
}

void AutonomyNode::setup()
{
  RCLCPP_INFO(get_logger(), "Setting up");
  create_behavior_tree();
  RCLCPP_INFO(get_logger(), "BT created");

  const auto timer_period = 500ms;
  timer_ = this->create_wall_timer(
      timer_period,
      std::bind(&AutonomyNode::update_behavior_tree, this));

  rclcpp::spin(shared_from_this());
  rclcpp::shutdown();
}

void AutonomyNode::create_behavior_tree()
{
  BT::BehaviorTreeFactory factory;

  // register bt node

  BT::NodeBuilder builder =
      [=](const std::string &name, const BT::NodeConfiguration &config)
  {
    return std::make_unique<GoToPose>(name, config, shared_from_this());
  };

  factory.registerBuilder<GoToPose>("GoToPose", builder);

  RCLCPP_INFO(get_logger(), bt_xml_dir.c_str());

  tree_ = factory.createTreeFromFile(bt_xml_dir + "/tree.xml");
  RCLCPP_INFO(get_logger(), "3");
}

void AutonomyNode::update_behavior_tree()
{
  BT::NodeStatus tree_status = tree_.tickRoot();

  if (tree_status == BT::NodeStatus::RUNNING)
  {
    return;
  }
  else if (tree_status == BT::NodeStatus::SUCCESS)
  {
    RCLCPP_INFO(this->get_logger(), "Finished Navigation");
  }
  else if (tree_status == BT::NodeStatus::FAILURE)
  {
    RCLCPP_INFO(this->get_logger(), "Navigation Failed");
    timer_->cancel();
  }
}

int main(int argc, char **argv)
{
  rclcpp::init(argc, argv);
  auto node = std::make_shared<AutonomyNode>("autonomy_node");
  node->setup();
```

```
74
75    return 0;
76  }
77
78
```

src\navigation_behaviors.cpp

```
1   #include "navigation_behaviors.h"
2   #include "yaml-cpp/yaml.h"
3   #include <string>
4
5   GoToPose::GoToPose(const std::string &name,
6                      const BT::NodeConfiguration &config,
7                      rclcpp::Node::SharedPtr node_ptr)
8       : BT::StatefulActionNode(name, config), node_ptr_(node_ptr)
9   {
10    action_client_ptr_ = rclcpp_action::create_client<NavigateToPose>(node_ptr_, "/navigate_to_pose");
11    done_flag_ = false;
12  }
13
14  BT::PortsList GoToPose::providedPorts()
15  {
16    return {BT::InputPort<std::string>("loc")};
17  }
18
19  BT::NodeStatus GoToPose::onStart()
20  {
21    // Get location key from port and read YAML file
22    BT::Optional<std::string> loc = getInput<std::string>("loc");
23    const std::string location_file = node_ptr_->get_parameter("location_file").as_string();
24
25    YAML::Node locations = YAML::LoadFile(location_file);
26
27    std::vector<float> pose = locations[loc.value()].as<std::vector<float>>();
28
29    // setup action client
30    auto send_goal_options = rclcpp_action::Client<NavigateToPose>::SendGoalOptions();
31    send_goal_options.result_callback = std::bind(&GoToPose::nav_to_pose_callback, this, std::placeholders::_1);
32
33    // make pose
34    auto goal_msg = NavigateToPose::Goal();
35    goal_msg.pose.header.frame_id = "map";
36    goal_msg.pose.pose.position.x = pose[0];
37    goal_msg.pose.pose.position.y = pose[1];
38
39    tf2::Quaternion q;
40    q.setRPY(0, 0, pose[2]);
41    q.normalize(); // todo: why?
42    goal_msg.pose.pose.orientation = tf2::toMsg(q);
43
44    // send pose
45    done_flag_ = false;
46    action_client_ptr_->async_send_goal(goal_msg, send_goal_options);
47    RCLCPP_INFO(node_ptr_->get_logger(), "Sent Goal to Nav2\n");
48    return BT::NodeStatus::RUNNING;
49  }
50
51  BT::NodeStatus GoToPose::onRunning()
52  {
53    if (done_flag_)
54    {
55      RCLCPP_INFO(node_ptr_->get_logger(), "[%s] Goal reached\n", this->name());
56      return BT::NodeStatus::SUCCESS;
57    }
58    else
59    {
60      return BT::NodeStatus::RUNNING;
61    }
62  }
63
64  void GoToPose::nav_to_pose_callback(const GoalHandleNav::WrappedResult &result)
65  {
66    // If there is a result, we consider navigation completed.
67    // bt_navigator only sends an empty message without status. Idk why though.
68
69    if (result.result)
70    {
71      done_flag_ = true;
72    }
73  }
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.8)
project(tb3_autonomy)

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()


# find dependencies
find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
find_package(rclcpp_action REQUIRED)
find_package(nav2_msgs REQUIRED)
find_package(behaviortree_cpp_v3 REQUIRED)
find_package(yaml-cpp REQUIRED)
find_package(tf2 REQUIRED)
find_package(tf2_geometry_msgs REQUIRED)

# Install directories
install(DIRECTORY
bt_xml launch
DESTINATION share/${PROJECT_NAME}
)

# Install C++ behaviors
set(BEHAVIOR_SOURCES
src/navigation_behaviors.cpp
)

set(TARGET_DEPENDS
    rclcpp
    behaviortree_cpp_v3
    yaml-cpp
    rclcpp_action
    nav2_msgs
    tf2
    tf2_geometry_msgs
)

include_directories(include)
add_executable(autonomy_node src/autonomy_node.cpp ${BEHAVIOR_SOURCES})
ament_target_dependencies(autonomy_node ${TARGET_DEPENDS})
target_link_libraries(autonomy_node ${YAML_CPP_LIBRARIES})

install(TARGETS
autonomy_node
DESTINATION lib/${PROJECT_NAME})


if(BUILD_TESTING)
  find_package(ament_lint_auto REQUIRED)
  # the following line skips the linter which checks for copyrights
  # uncomment the line when a copyright and license is not present in all source files
  #set(ament_cmake_copyright_FOUND TRUE)
  # the following line skips cpplint (only works in a git repo)
  # uncomment the line when this package is not in a git repo
  #set(ament_cmake_cpplint_FOUND TRUE)
  ament_lint_auto_find_test_dependencies()
endif()

ament_package()
```