Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Irányítástechnikai és Informatikai Tanszék

Izsó András

MOBILIS ROBOT NAVIGÁCIÓS STRATÉGIÁJÁNAK FEJLESZTÉSE ISMERETLEN

KONZULENS

Dr. Harmati István

BUDAPEST, 2021

Tartalomjegyzék

Összefoglaló

A robotika napjainkban tapasztalható rohamos fejlődése egyre több és egyre bonyolultabb feladatok végrehajtását teszik lehetővé. Az új alkalmazások nagy része a mobilis robotokhoz kötődik. Az irányítási feladat alsóbb szintjén ezért kulcsfontosságú szerepet játszik a robot pontos navigációja. A mobilis robotrendszerek egyik érdekes alkalmazási területe az ismeretlen területek feltérképezése, amely fontos szerepet kaphat természeti katasztrófák (pl. földrengés romjai közötti) életmentési feladataiban is.

Dolgozatomban Latombe és González-Baños 2002-es cikkében [1] leírt általános algoritmusát implementáltam és egészítettem ki, a működéshez szükséges konkrét algoritmusokkal, majd kísérletet tettem a továbbfejlesztésére.

A cikk egy általános algoritmust ír le feltérképezési feladatokhoz. Egy módszert dolgoztak ki, mellyel meghatározható, hogy honnan érdemes a következő mérést elkészíteni, hogy a legtöbb újonnan felfedezett területre számíthassunk. Munkám során az algoritmust először MATLAB környezetben implementáltam. Ezután a fizikai megvalósításhoz egy saját készítésű robotot terveztem. Az áramkör megtervezése után CAD program segítségével megterveztem a robot vázát is. Az így elkészített tervet a szimulációba importálva meggyőződtem arról, hogy a tervezett robot megfelelően teljesíteni tudja a követelményeket és megkezdtem a hiányzó lépések implementálását.

A pályatervezéshez a Rapidly Exploding Random Tree (RRT) algoritmust implementáltam. A mérési pontokban készített lokális térképek összefűzéséhez saját algoritmust dolgoztam ki, először az egymást metsző élek irányát felhasználva, majd megkíséreltem a problémát valószínűségi alapokra helyezve vizsgálni, mivel a valós szenzoroknál elkerülhetetlen a mérési zaj.

Az algoritmus valóságba történő átültetéséhez házilag legyártottam a robotot. A teszt összeállításban elhelyeztem egy kamerát is, amely felülnézetből látja a robotot. Ezután egy algoritmust implementáltam, amely ArUco marker segítségével tudja meghatározni a robot helyét és helyzetét, ezzel globális és meglehetősen pontos alapot szolgáltatva a navigációhoz. A fejlesztés könnyítése érdekében egy grafikus felületet is összeállítottam, ahol folyamatosan nyomon követhetjük a feltérképezés állapotát és a robot további jellemzőit.

Abstract

The rapid development of robotics of these days, makes the execution of more and more complex tasks possible. Most of the newer applications are related to mobile robots. On the lower level of the control problem therefore the exact navigation of the robot plays an essential role. Exploration of an unknown environment is an interesting application of robotic systems, which can get a crucial role in the life saving missions after environmental disasters (e.g.: between ruins of an earthquake).

Through my paper I implement and extend the general algorithm of Latombe and González-Baños [1] from 2002 with concrete steps which are required for a functioning system. Then I attempt to develop it further.

The article describes a general algorithm for exploration problems, which can determine which is the best location for the next measurement, in the sense of the most newly explored area expectations. First, I have implemented the algorithm in a MATLAB environment. Then I have designed a robot for the physical implementation. After designing the electric circuit, I have also designed the chassis of the robot using a CAD software. Using the finished 3D design, I could verify in a simulation that the robot is meeting the requirements. The following step was the implementation of the missing parts.

For the path planning I have implemented a Rapidly Exploring Random Tree (RRT) algorithm. I have developed my own strategy for the merging of the local maps, prepared in different measurement points. At first, I considered the direction of the intersecting edges, then I have tried to take the problem on probability basis, since by using real sensors the measurement noise is inevitable.

The algorithm was to be tested on my home-made robot. In the test set up, I have also placed a camera, which looks at the robot from the ceiling. I have implemented an algorithm, which is able to detect the position and orientation of the robot, using ArUco markers. This provides a precise base for the navigation. To make the development easier, I have also pieced together a graphical user interface, on which we can follow the state of the mapping and other traits of the robot.

1 Bevezetés

1.1 Dolgozatom célja

A munkám célja egy olyan rendszer létrehozása volt, melyen a szimuláció mellett a valóságban is tesztelhetőek különböző navigációs és feltérképező algoritmusok. Az elsődleges feladat az egyágenses működés megvalósítása volt, viszont a fejlesztés során végig szem előtt tartottam, hogy a rendszer minél könnyebben kiterjeszthető legyen több ágens kezelésére. A rendszerhez egy Latmbe és González-Baños által kidolgozott, következő mérési pontot meghatározó módszerét [1] is implementáltam, valamint kiegészítettem teljes feltérképező algoritmussá.

Ehhez először egy robotot terveztem, mely rendelkezik a szükséges szenzorokkal és beavatkozó szervekkel a feladat végrehajtásában. A tervezés során ügyeltem arra, hogy olcsón, otthon is legyártható legyen. Ezután MATLAB környezetben létrehoztam egy szimulációs környezetet az algoritmus implementálására és tesztelésére, a Webots robot szimulátor segítségével. Miután a szimulátorban meggyőződtem az algoritmus működőképességéről és a robot alkalmasságáról, a korábban tervezett és a tapasztalatok alapján módosított robotot otthon elkészítettem és a beágyazott rendszeren is implementáltam az algoritmust. Emellett a sikeres működéshez egy kamerakép alapú pozíciómeghatározó programot is készítettem. Az összeállított rendszert fizikai környezetben is kipróbáltam.

Dolgozatomban először hasonló munkák áttekintésével vezetem be a feltérképező algoritmusok mai állásába az olvasót, ezek közül az implementált Next Best View algoritmust részletesebben is bemutatom. Ezután áttekintem a robotom felépítését, milyen komponensekből épül fel és miért ezeket választottam, majd rátérek a szimuláció felépítésére, az általam implementált algoritmusok működésére, az általam kitaláltakat részletesebben taglalva. Végül ismertetem a fizikai környezetben megvalósított rendszer felépítését, működését. Dolgozatomat a tapasztalatok összegzésével és a további célok kitűzésével zárom.

1.2 Hasonló munkák

A hardvertechnológia fejlődésével egyre több lehetőség van növekvő számítási kapacitással és komplexebb szenzoros felszereltséggel rendelkező mobilis robotok

fejlesztésére. Az elmúlt 20 évben ezzel együtt egyre nagyobb figyelmet kaptak a különböző feltérképező és navigáló algoritmusok, melyet Panigrahi [2] foglalt össze. A hagyományosabbnak tekinthető kemény számítási módszerek mellett, mint például a Kálmán-szűrő, egyre nagyobb teret nyernek a különböző tanuló algoritmusok, optimalizáló eljárások, az úgynevezett lágy számítási módszerek, valamint a gépi látás.

A robot pozíciójának meghatározásának (lokalizációjának) feladatkörét három nagy csoportra bonthatjuk. Az első típusa, amikor a robot ismeri a kezdeti pozícióját, és ezután szeretnénk követni a mozgását és minden időpillanatban tudni, hogy hol van. Ez a feladat hatékonyan és precízen megoldható a Kálmán-szűrővel, nemlineáris esetben valamelyik kiterjesztésével. Ezeknek a módszereknek viszont szükségük van a robot korábbi (kezdeti) pozíciójának ismeretére. Amennyiben a robot pozíciója ismeretlenné válik (a robot eltéved), akkor pusztán ezekkel az algoritmusokkal nem lokalizálható újra. Itt kapcsolódik a lokalizáció második fajtája, a globális lokalizáció, mely során a robot nem rendelkezik kezdeti információval a hollétéről, viszont a környezeten végzett mérések alapján felismeri, képes lokalizálni magát. Ez a probléma kezelhető Markov-modell használatával (a Kálmán-szűrő ennek egy speciális esete), valamint az újabban megjelent, egyre nagyobb teljesítményű kártyáknak köszönhetően (pl.: Raspberry PI, Jetson Nano) tanuló, optimum kereső eljárásokkal. A problémát dinamikus optimumkereséssé alakítva többféle ilyen algoritmust is sikeresen alkalmaztak, például részecske raj optimalizációt, genetikus algoritmusokat, fuzzy rendszereket, vagy ezek kombinációját egymással, vagy az egyszerűbb követő algoritmusokkal kombinálva [2] [3]. A lokalizációs problémakör harmadik nagy csoportja az, amikor a robot teljesen ismeretlen környezetbe kerül. Ez az egyszerre lokalizáció és feltérképezés (Simultaneous Localization And Mapping – SLAM). Itt a nehézséget az adja, hogy lokalizálni a mért adatok térképhez való illesztéssel tudnánk, térkép építéséhez viszont szükség lenne a pozíció pontos ismeretére. Erre a különböző, jellemző illesztés alapú (pattern matching) algoritmusok adnak megoldást, melyek minden mintavételben jellegzetes pontokat keresnek, és az előző mintavételhez illesztve határozzák meg a robot elmozdulását. A LIDAR mellett egyre gyakrabban és hatékonyabban alkalmaznak gépi látás alapú módszereket [4] [5] [6].

A mellett, hogy a növekvő számítási kapacitást tanuló, illetve gépi látás alapú algoritmusokkal aknázzák ki, a másik fejlesztési irány a többágenses rendszerek alkalmazása [7]. Ezek a rendszerek több robot együttműködését próbálják minél

optimálisabban megvalósítani, mivel így jóval hatékonyabb működés érhető el. Itt megkülönböztethetünk központi és elosztott működést. Előbbit egyszerűbb lehet megvalósítani, viszont utóbbi sokkal robusztusabb működésre képes. Természetesen multiágensű megvalósításoknál is felhasználják a tanuló algoritmusok új eredményeit [8].

Amellett, hogy az ilyen tanuló algoritmusokkal, és összetettebb szenzorokkal ellátott robotok rendkívül hatékonyan tudják a feladatukat végrehajtani, ezeknek az anyagi vonzata is jelentős, így nem is minden kutató csoport engedheti meg maguknak a használatukat, valamint termékké fejlesztve is magas áruk lenne. Így relevanciája van az olyan fejlesztéseknek is, melyek olcsóbb, egyszerűbb szenzorokat használva próbálnak minél hatékonyabb működést elérni [9] [10].

2 Implementálandó algoritmus ismertetése

Munkám során a Latombe és González-Baños által leírt Next Best View algoritmust [1] implementáltam és egészítettem ki a szükséges lépésekkel, valamint próbáltam hatékonyabbá tenni a működést. Az alábbiakban ismeretem ezt az algoritmust. Az én implementációmhoz kötődő leírás a Szimuláció fejezetben található.

2.1 Next Best View algoritmus

A Next Best View (NBV) algoritmus arra ad egy eljárást, hogy ismeretlen környezet feltérképezésekor hogyan határozható meg, hogy honnan érdemes a következő megfigyelést elvégezni. Ehhez egy LIDAR jellegű szenzort feltételez, amely 360°-ban, egyenletes közönként méri meg a tárgyak tőle vett távolságát, és ezeket a méréseket rendezett sorban szolgáltatja.

Az algoritmus a környezetet egy zárt görbével határolt területnek feltételezi, amelyen belül nincs egyéb akadály. A kerület egy pontjának láthatóságát a környezet egy pontjából három feltételhez köti:

1. A két pont között vezető szakasz nem metszi a környezet határát
2. A két pont távolsága nem halad meg egy határértéket (a szenzor mérési tartományát)
3. A két pontot összekötő egyenes, és a határoló görbe normálisa a vizsgált pontban egy megadott szögnél (τ) kisebb szöget zár be

Az algoritmus a biztonságos régió kiterjesztésére törekszik. Biztonságos régió az, amiről biztosan tudjuk, hogy ott nincs akadály (tehát a már felfedezett terület). A mérés alapján két élcsoportot különböztet meg. A kötött élek (solid edge) azok, amelyeket a szenzor észlelt, tehát ott biztosan van akadály. Szabad élek azok az élek, ahol a szenzor nem észlelt akadályt, viszont ezeken túl a láthatóság feltételei nem teljesülnek, és emiatt nem lehetünk biztosak abban, hogy szabad az út, vagy pont akadály következik.

A tanulmány során bebizonyították, hogy a szabad ívek maximum három részből állnak, melyek mindegyike egy szakasz, egy körív (melynek sugara a szenzor hatótávja), vagy egy exponenciális görbe, amely az alábbi képlettel adódik:

$$r = r_0 \cdot e^{\pm\lambda\theta} \tag{1}$$

Ahol r_0 az ív kiinduló pontja (a megelőző kötött él utolsó pontja vagy a következő első pontja), λ=tan(τ), és az előjel attól függ, hogy az ív az óramutató járásával

megegyező, vagy ellentétes irányba kanyarodik. Utóbbi eset a szabad élet megelőző kötött él utolsó pontjából induló görbe esetén áll fent, korábbi pedig a következő kötött él első pontjából induló görbe esetén áll fent.

A környezet egy pontjából végzett mérés esetén, a kötött és szabad görbékből egy zárt görbe áll elő, amely a mérés során feltérképezett biztonságos területet jelenti, ez a lokális térkép. A lokális térképet ezután illeszteni kell a globális térképhez (a már korábban felfedezett biztonságos régióhoz), és képezni az uniójukat, ezzel előállítva az új globális térképet.

A globális térkép alapján ezután az algoritmus azt a pontot próbálja meghatározni, ahonnan mérést végezve, a lehető legnagyobb területet felfedezésére számíthatunk. Ezekhez véletlen pontokat választ, az alábbi módon:

1. Véletlen pontok választása a szabad élek láthatósági tartományából
2. Minden pontra megvizsgálja, hogy a kötött élek milyen hosszan látszanak, ha kisebb, mint az illesztő algoritmus által meghatározott határérték, akkor elveti a pontot
3. Megvizsgálja minden pontra, hogy elérhető-e a robot jelenlegi pozíciójából, ha nem akkor elveti.

   A kiválasztott pontokra az alábbi képlettel számol egy pontszámot

$$g(q) = A(q) \cdot e^{-\lambda L(q)} \tag{2}$$

ahol λ egy konstans, L(q) a pontba vezető út hossza, A(q) pedig a pontból potenciálisan felfedezhető új terület mérete.

Lambda határozza meg, hogy az algoritmus a nagy, új felfedezhető területeket (nagy λ), vagy a kis, közel lévő kiegészítéseket részesítse előnyben (kis λ).

Innentől kezdve az algoritmus iteratívan folytatható, egy kilépési feltételig. Ez a kilépési feltétel célszerűen az lenne, hogy a térkép már nem tartalmaz szabad éleket, mivel ez azt jelent, hogy a teljes határoló görbét felfedeztük. A gyakorlatban ez nehezen teljesíthető, ezért jobb választás lehet az, ha az összes szabad él hossza egy előre megadott határ alá csökken.

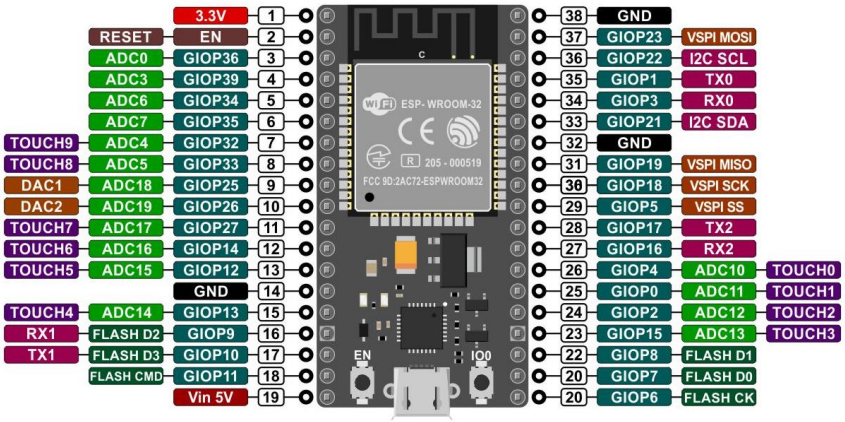
3 A robot hardveres megvalósítása

A robot megvalósításában az egyik kulcs elem az omnidirekcionális meghajtás volt. Ennek több előnye van, melyeket a Szimuláció fejezetben részletezek, a legfontosabb, hogy előzetes manőver nélkül képes bármilyen irányba elmozdulni. A felhasznált eszközökben fontos korlátozó tényező volt az ár, mivel önerőből tudtam megvalósítani, és egy robot csapathoz további példányok elkészítése is szükséges. Továbbá követelmény volt, hogy képes legyen vezeték nélküli kommunikációra kliens alkalmazással és önálló navigációra.

3.1 Kiválasztott alkatrészek

3.1.1 Mikrovezérlő

A robot irányításához az ESP-32 mikrokontrollert (1. ábra) választottam, elsősorban a natív WiFi kezelése és a renget elérhető forrást miatt. A mikrokontroller rendelkezik 2db, kifejezetten DC motorok meghajtásához használt pulzusszélesség modulált (pulse width modulation – PWM) jel előállítására képes hardver elemmel (motor control pulse width modulator – MCPWM), melyek egyenként 3 motor számára képesek vezérlő jeleket előállítani. Ezen kívül természetesen rendelkezik I2C és SPI interfésszel, ami a szenzorokkal való kommunikációhoz szükséges, valamint 34db GPIO lábbal, melyekhez 20db ADC köthető, így sokféle kiegészítő alkalmazása lehetséges (visszajelző LED-ek, gombok), amik a fejlesztést és hibakeresést könnyíthetik.



1. ábra Az ESP-32 fejlesztőkártya és kivezetései

3.1.2 Inerciális mérő egység

Az autonóm robotok és járművek navigációjához sztenderdnek mondható az inerciális mérőegység (inertial measurement unit – IMU). Ezek a szenzorok általában egy tokban tartalmaznak 3 tengelyes gyorsulásmérőt, giroszkópot és magnetométert, lehetővé téve a különböző mozgások követését [11]. Választásom az MPU-9250 típusra (2. ábra) esett, mivel könnyű és olcsó beszerezhetősége mellett megfelelő pontossággal bír, mintavételi frekvenciája, méréstartománya konfigurálható, valamint digitális aluláteresztő szűrővel is rendelkezik, amely a lassabb mozgásoknál a mérési zajt jelentősen csökkenti. A mikrokontrollerhez I2C interfészen keresztül tud csatlakozni, és elérhető hozzá megbízható kezelő könyvtár, így a fejlesztést gyorsítandó, nem nekem kellet ezzel foglalkozni.

2. ábra MPU-9250, 3 tengelyes gyorsulásmérő, giroszkóp és magentométer

3.1.3 Távolságmérő

A kétdimenziós feltérképezési feladatokhoz gyakran használt eszköz a LIDAR (Light Detection and Ranging) szenzo [5]. Egy ilyen eszköz beszerzését viszont pénzügyileg nem engedhettem meg magamnak, így más módszerrel kellett a feltérképezést megvalósítani.

Kihasználva, hogy a robot könnyen tud a saját tengelye körül forogni, a LIDAR emulálására egy távolságmérő mellett döntöttem. Ebből két nagy kategória létezik, az ultrahangos és a lézeres távolságmérők. A lézeresek pontosabbak, megbízhatóbbak és kevésbé okoz nekik gondot, ha nem merőleges felülettel állnak szemben, így ezt a fajtát választottam, ezen belül is a VL53L0X típust (3. ábra). Az átlátszó felületek (pl.: üveg, plexi) gondot okozhatnak a szenzornak, ennek tudatában kell a környezetet megválasztani.

A választott típus 30-2000mm közötti méréstartománya ideálissá teszi beltéri feltérképező algoritmusokhoz. I2C buszon keresztül konfigurálható, így nem foglal

további lábat a mikrokontrolleren. A kezeléséhez szintén elérhető volt függvény könyvtár.

3. ábra GY-52 VL53L0X lézeres távolságszenzor

3.1.4 DC motor

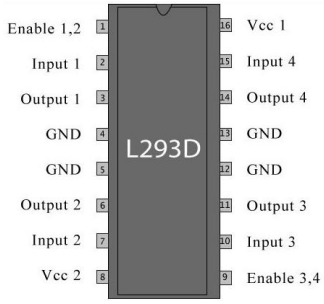
A robothoz enkóderrel felszerelt DC motort kerestem, 12V tápfeszültséghez, mivel a tapasztalataim szerint ezek azonos képességek mellett olcsóbbak más feszültségszinthez (pl.: 6V, 24V) készített társaiknál, viszont a feszültségszint könnyen előállítható például 3 cellás Lithium-Polimer (LiPo) akkumulátorral.

A választott GA12-N20 motor mini DC motort (4. ábra) választottam 1:150 áttétellel. Ez kis méretben tartalmazza az elvárásokat, enkóder 11 pulzus/fordulattal generál pulzusokat, amely a kiválasztott áttétellel és kerékkel 50nm elmozdulás felbontást jelent. Az áttétel kiválasztásánál inkább a nagyobb nyomatékot és áttételt, semmint a nagy fordulatszámot tartottam szemelőtt, mivel a robottól nem elvárás a gyorsaság, ellenben a pontosság igen. A kiválasztott motor 3kg*cm nyomatékkel rendelkezik, ez elegendő a robot mozgatásához.

4. ábra GA12-N20 mini DC motor, enkóderrel

3.1.5 Motorvezérlő

Motorvezérlőnek az L293D dupla H-hidas típust (5. ábra) választottam. Ezzel 2db DC motor hajtható meg, 4,5-36V tápfeszültséget támogat, valamint 1,2A folyamatos áramot bír, így megfelel az alkalmazásnak (a motorok kevesebb mint 0,5A maximális áramfelvétellel rendelkeznek).



5. ábra L293D H-hidas motorvezérlő

3.1.6 Kerék

Az omnidirekcionális meghajtáshoz kétféle speciális kerék közül lehet választani, ami a pontos kialakítástól függően. Én a három kerekes struktúrát alkalmaztam, mivel annak kisebb alkatrészigénye van (részletek a robot kinematikai leírásánál 4.2). Ehhez az úgynevezett svéd-kerékre (6 ábra6 ábra A robot meghajtásáho használt omnidirekcionális kerék) van szükség, mely kerületén kis görgők találhatóak. Ezek lehetővé teszik, hogy a robot a kerékre merőleges irányba is könnyen el tudjon mozdulni.

6 ábra A robot meghajtásáho használt omnidirekcionális kerék

3.1.7 Akkumulátor

A tápellátás biztosításához egy 2600mAh kapacitású, 3S kapcsolású Lítium-Polimer (LiPo) akkumulátort választottam, mely így maximum 12.9V feszültséget biztosít. A LiPo akkumulátorok előnye, hogy a többi típushoz képest nagyobb áramot szolgáltat, és kisebb a mérete. Hátránya, hogy sokkal érzékenyebb a mélykisülésre és a

túltöltésre. Ezek elkerülésére beszereztem hozzá egy LiPo őrt is, amely jelez ha ilyen probléma merülne fel és lekapcsolja az áramkört.

3.2 Elektromos kapcsolás

Mivel prototípusról van szó, a robot nyákjához szenzor és alkatrész modulokat használtam, nem egy lapra terveztem az egészet, ezzel könnyebben cserélhetőek az esetleges meghibásodott, vagy mégsem megfelelőnek ítélt alkatrészek.

Az áramkör tervezésénél (8 ábra) a fő szempont az otthoni gyárthatóság lehetővé tétele volt. Ehhez a szükségesnél vastagabb vezetékezést használtam és az elrendezésnél szem előtt tartottam, hogy minél kevesebb vezeték fusson a nyomtatott áramkör (NYÁK) hátoldalán (7 ábra). Ezen kívül a tervet úgy alakítottam ki, hogy a robot 3D nyomtatásban elkészülő vázára kényelmesen illeszkedjen.

A tápfeszültséget a bekötéstől egy kapcsolón keresztül vezettem a motorvezérlőkhöz, hogy a robot biztosan mozgásképtelenné tehető legyen.. A motorokhoz vezető táp- illetve föld vezetékek elvezetésével ügyeltem arra, hogy a motorokon átfolyó nagyobb áram a mikrokontrolleren és a szenzorokon ne folyjék át, ezzel elkerülve a potenciális meghibásodásokat.

A tápfeszültség és a föld közé egy 100kΩ/20kΩ méretű feszültségosztót terveztem, melynek a kimenetét a mikrokontroller egy analóg bemeneti lábára vezetve nyomon követhető az akkumulátor töltöttségi szintje is.

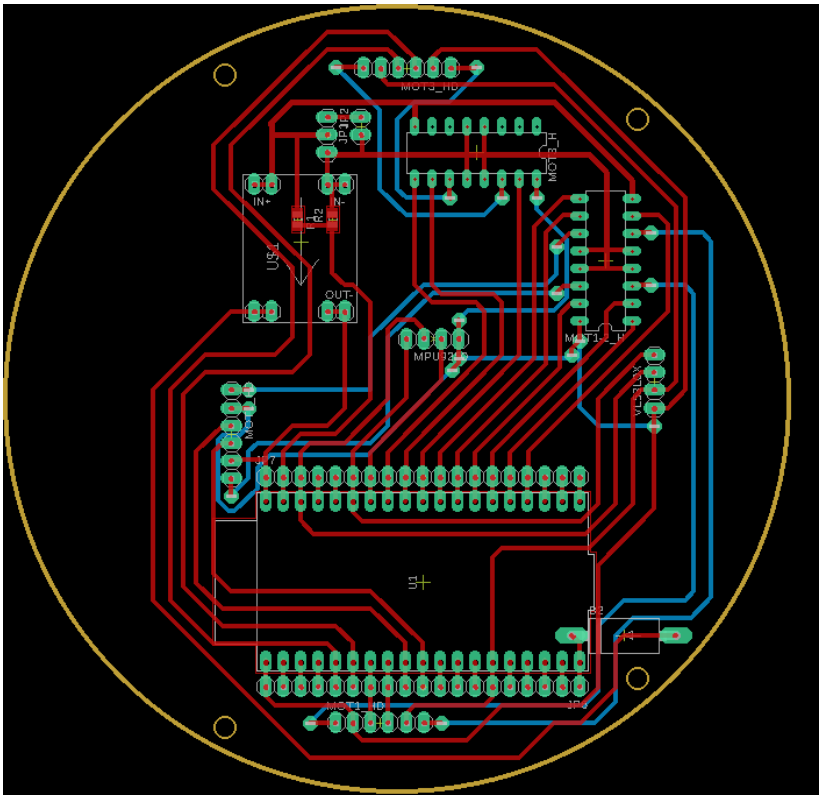
Az 5V-os feszültség szintet egy DSN-1504 DC-DC buck konverter segítségével állítom elő a tápfeszültségből. Ebből az ESP beépített feszültségstabilizátora állítja elő a saját maga és a szenzorok számára szükséges 3,3V-ot. A mikrokontrollert egy dióda segítségével választottam el a 12V-os áramkörtől. Erre azért van szükség, mert ha a mikrokontroller USB vezetéken keresztül számítógépre van csatlakoztatva, előfordulhat, hogy a buck konverteren visszafelé áram folyik. A motorok így a számítógép USB csatlakozóján keresztül túl nagy áramot vennének fel, ami kárt is okozhat, de a munkát is nehezítené.

Az alkatrészek elhelyezésénél a cél az volt, hogy a lézeres távolságmérő megfelelő irányban, középen helyezkedjen el, úgy, hogy a mérési tartomány minél jobban kihasználható legyen (tehát kb 3cm-re a robot peremétől). Ezen kívül a magnetométert

próbáltam a robot forgástengelyéhez minél közelebb elhelyezni, valamint a motorcsatlakozókat is úgy, hogy minél kevesebb vezetékezésre legyen utólag szükség.

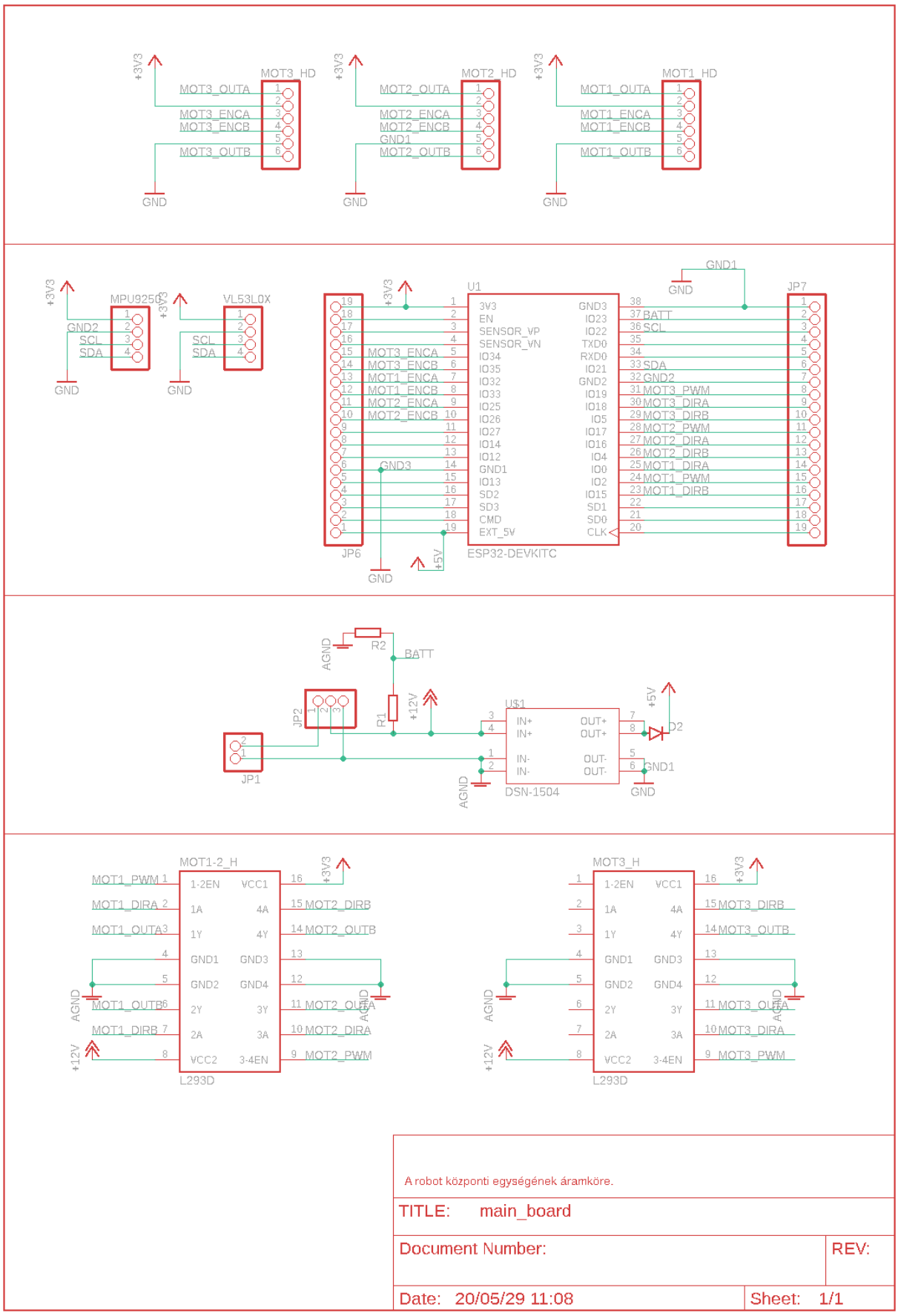
A mikrokontroller lábkiosztása szerencsére rugalmasan konfigurálható, így itt az alkatrészek fizikai elhelyezkedése alapján választottam meg a csatlakozásokat.

A NYÁK lap tervezése során az ESP csatlakozóinak kivezetésével biztosítottam, hogy a robot rugalmasan bővíthető legyen további egységekkel.



7 ábra A megtervezett nyák vezetékezése. A cél az otthoni gyárthatóság és a vázra való illeszkedés volt

A megtervezett elkészítését otthon végeztem, a felmerülő gyártási költségek elkerülésére, a szerelés könnyítése miatt nem rendelkezik sok visszajelző LED-del. Az elkészítéséhez fényes papírra nyomtatva a tervet rezezett lemezre vasaltam át, majd Nátrium-Perszulfátot használva marattam le.



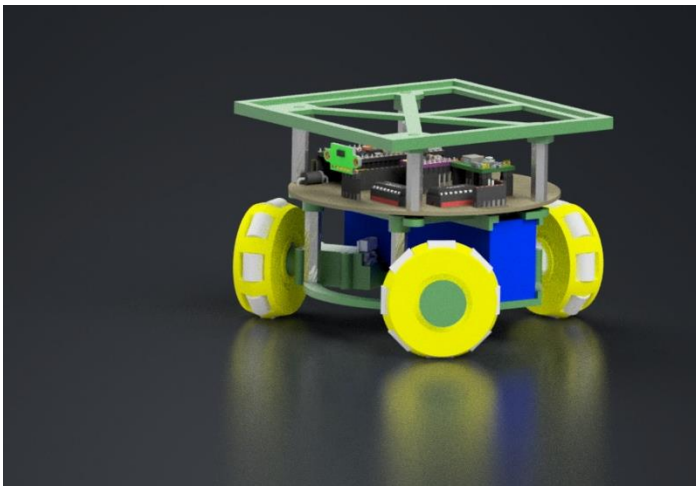
8 ábra A NYÁK lap kapcsolásai rajza

3.3 Váz

A robothoz a korábbi tanszéki robotok alapján terveztem, hogy adott esetben azokkal könnyen együttműködhessen. Az alkatrészek kialakításánál figyelembe vettem a 3D nyomtatás technológiai korlátait és úgy terveztem meg, hogy gyártható legyen támaszték nélkül.

Az omnidirekcionális kialakításból fakadóan adta magát a kör alaprajz. Ezen egy akkumulátortartót alakítottam ki, valamint a motoroknak kis zsebeket. Készítettem egy illesztő panelt a NYÁK lemezhez is, a robot tetejére pedig egy platformot készítettem, amely a marker tartására szolgál. A kerekeket egy illesztő darabbal fogattam hozzá a motor tengelyéhez, hernyó csavar segítségével.

A váz tervezéséhez az alkatrészek CAD modelljeit is felhasználtam a jó illeszkedés érdekében. Az így összeépített robot a később megvalósított szimulációhoz is hasznos volt, mivel azt a tényleges robot modellel paramétereivel végezhettem. A megtervezett robotot a 9 ábra mutatja be.



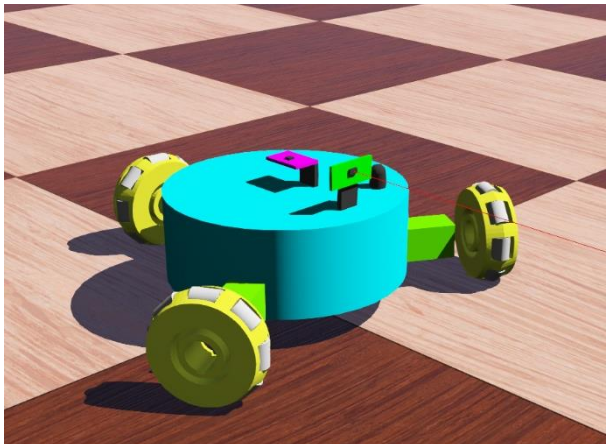
9 ábra A robotról készített látványterv

4 Szimuláció

Mielőtt a gyakorlatban is legyárottam és összeszereltem a robotot, szerettem volna meggyőződni az algoritmus és a robot működéséről szimulációban. Ezen kívül az algoritmus tesztelése is egyszerűbb itt, több adat áll rendelkezésünkre, mint a valóságban (pl.: pontosan ismerjük a robot pozícióját), elhagyhatunk pár nehezítő tényezőt (pl.: súrlódás). Ehhez először MATLAB környezetben implementáltam az egyes lépéseket.

4.1 Környezet ismertetése

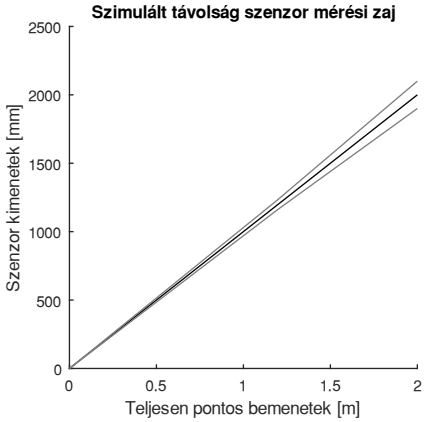
A szimulációs környezetnek eredetileg MATLAB-Simulinket szántam, de a Simulink sok megkötése az adatok dimenziójára vonatkozóan rendkívül megnehezítette a haladást. Ezért váltottam a Webots szimulátorra. Ennek a kezelése is sokkal könnyebb, kiforrottam, mint a Simulink 3D Animation toolboxa. Lehetőségünk van egy saját robotot is importálni, felszenzorozni, beavatkozó szervekkel ellátni és a működését vizsgálni az általunk meghatározott fizikai pontossággal (eldönthetjük mit modellezünk és mit nem). A környezet többek között a MATLAB nyelvet is támogatja, így az átállás sem okozott nagy problémát. További pozitívuma, hogy a MATLAB többi funkciója – gondolok itt elsősorban a plot mechanizmusokra – közvetlen elérhető, így az adatok könnyen megjeleníthetőek, amit ki is használtam. Ahogy azt a 10 ábra is mutatja, a robotot nem teljes pontossággal modelleztem, mivel a kis elketronikai alkatrészek például nem sokban módosítják a viselkedését, viszont a számolást nagyban nehezítik. Ezzel szemben a



10 ábra A Webots szimulátorban használt, korábbi robot modell. A kerekeket sikerült beljebb hozni, viszont az itt használt modellt már felesleges lett volna hozzá igazítani

kerekeket pontosan lemásoltam, a kis görgőkkel együtt, hogy lehetőleg valósághű viselkedést kapjak.

A szimulátor lehetőséget biztosít a szenzorokhoz zajmodell felvételére, a várható érték és szórás megadásával, lookup table megoldással. A pontok között a program lineáris interpolációval számol. A szenzor adatlapja alapján 1,2m távolságon 3%, 2m távolságon 5% pontosságot írt, alapján állítottam be a szimulációban a zaj mértékét. Ennek a jellegét a 11 ábra szemlélteti.



11 ábra A szenzor adatlapja alapján készített szenzor zaj leírás. Fekete vonal a nominális érték, szürkével a szórás határok

A feltérképezéshez szükséges algoritmusokat X-Y síkban végeztem el, úgy, hogy az 0 orientáció az X tengely irányába mutat (a megszokott módon). A megjelenítést és a navigációt nem akartam bonyolítani más koordináta rendszerek bevezetésével (pl.: a járműveknél szokásos North-East-Down (NED) keret), továbbá a robot csak síkban mozog, így nincs igazán jelentősége. A megjelenítés könnyítése és az intuíció megtartása érdekében a térképet úgy reprezentálom, hogy az Y tengely mutat felfele („észak felé”), az X tengely pedig jobbra („kelet felé”). A robothoz rögzített koordináta keretet ehhez igazodva úgy választottam meg, hogy az X tengely mutat előre, a távolságszenzor irányába, az Y tengely pedig balra.

4.2 A robot kinematikai modellje és irányítása

Kinematikai modell

Az omnidirekcionális kialakítás több módon is elérhető. Ezeke kerekek számában, helyében és helyzetében, valamint a rajtuk található görgők irányában különböznek. Mivel a robot mozgását nem korlátozza semmi, bármilyen irányú sebességgel el tud

indulni, szerintem érdemes a robot irányítását globális koordináta keretre építeni, és utána a kapott beavatkozójel vektort transzformálni először a robothoz rögzített keretbe, majd az egyes kerekekre szétbontani. Így magas szinten a robot mozgása rendkívül egyszerűen tervezhető, a DC motorok szabályzására pedig mára már jól bevált módszerek léteznek. A globális sebességvektort kezelve bemenetként, az alábbi, rendkívül egyszerű állapotegyenlet adódik:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} v_x \\ v_y \\ \omega \end{pmatrix}$$

Ahol x és y a robot pozícióját, θ a robot orientációját, v_x v_y az egyes koordináta keretek menti sebességet, ω pedig a robot szögsebességét jelöli. Előbbiek alkotják a robot állapotvektorát, utóbbiak a bemeneti jeleket.

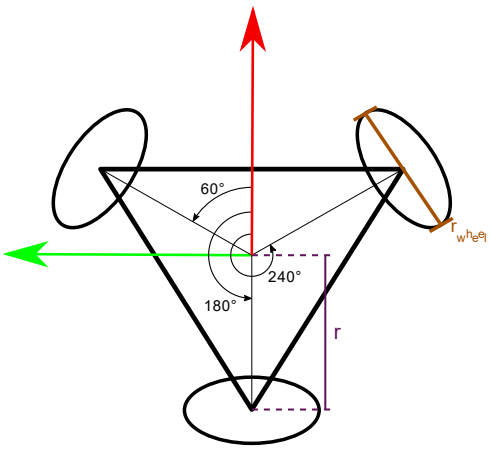
A fenti modellhez előállított sebességvektorra először egy elforgatást alkalmazok, amivel a robothoz rögzített, lokális koordináta keretbe transzformáljam:

$$\begin{pmatrix} v_{x|robot} \\ v_{y|robot} \\ \omega_{robot} \end{pmatrix} = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} v_x \\ v_y \\ \omega \end{pmatrix}$$

A kerék forgási sebességekre bontás ezután általánosan levezethető. Az én esetemben az alábbi transzformációra egyszerűsödik [12]:

$$\begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \frac{1}{r_{wheel}} \begin{bmatrix} 1 & 0 & -r \\ -\frac{1}{2} & -\sin\left(\frac{\pi}{3}\right) & -r \\ -\frac{1}{2} & \sin\left(\frac{\pi}{3}\right) & -r \end{bmatrix} \begin{pmatrix} v_{x|robot} \\ v_{y|robot} \\ \omega_{robot} \end{pmatrix}$$

Ahol r a kerék távolsága a robottól, r_{wheel} pedig a kerék sugara. A robot vázlatát a 12 ábra szemlélteti.

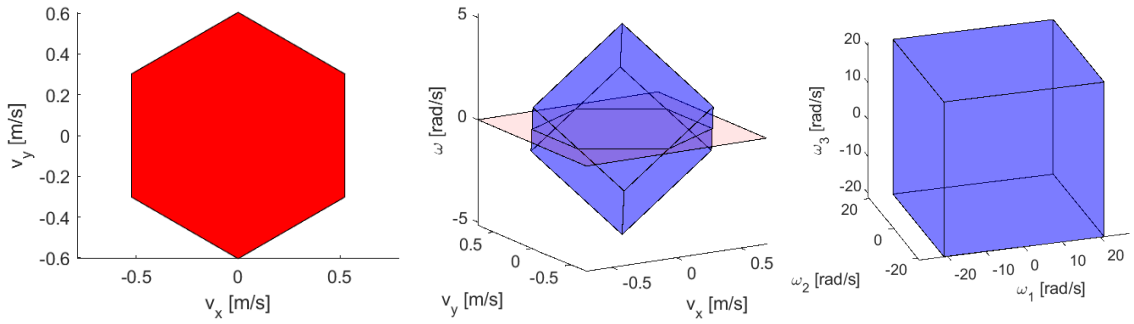


12 ábra A robot sematikus felépítése. A kerekek 120°-onként vannak elhelyezve, r távolságra a robot középpontjától és r_{wheel} átmérőjűek

Pályakövetés, szabályzás, szaturáció

Az ütközésmentes pályatervező algoritmusok a pályát egy töröttvonal formájában szolgáltatják, így a pályakövetést is ilyen formára alakítottam. A robot kialakításának köszönhetően ez szakaszonként egy 1 dimenziós pozíció szabályzásra egyszerűsödik. Ez egy P szabályzóval hatékonyan megoldható. A szimulátor lehetőséget biztosít a kerekek sebességének ugrásszerű változtatására, így ezzel itt nem foglalkoztam, a valós rendszeren történő implementáláskor terveztem hozzá szabályzót, melyet a 5.2.1 fejezetben részletezek.

A kerekek maximális forgási sebessége korlátozva van a szimulációban (ahogy az a valóságban is van). Ezzel foglalkozni kell, mert előfordulhatna a helyzet, hogy a szabályzó olyan beavatkozó jelet kér, amit csak az egyik kerék nem tud teljesíteni, így a robot teljesen más irányba mozog, mint azt elvárjuk. A szaturáció kezelése a v_x, v_y és ω alkotta térben körülményes lenne, mivel itt a kiadható jelek halmaza egy csúcsára állított romboéderként jelenik meg, ahogy azt a13 ábra is mutatja. Ennek a vizsgálata komplikált lenne, így a szaturációt a kerekek forgási sebessége által alkotott térben vizsgálom. Itt könnyen ellenőrizhető és értelmezhető megszorításokat tudunk adni. Abban az esetben, ha valamelyik kerékre a kikért beavatkozó jel túllépné a lehetséges maximumot, mindhárom kerék sebességét arányosan csökkentem úgy, hogy a legnagyobb is beleférjen a korlátok közé. Ennek eredményeképpen lassabban fog haladni a robot, viszont nem tér le a kívánt irányról, tehát az ebből fakadó ütközések elkerülhetőek.



13 ábra A kiadható beavatkozó jelek halmaza. Balról jobbra: a kiadható sebességek, 0 forgási sebesség esetén, ugyanez a forgási sebességet is ábrázolva a függőleges tengelyen, végül a kiadható beavatkozó jelek a kerék sebességek terében ábrázolva.

4.3 Feltérképező algoritmus megvalósítása

Az NBV algoritmus megad egy keretet, hogy hogyan érdemes a feltérképezést végezni, milyen fizikai korlátokra kell a biztonságos régió előállításakor ügyelni. Szabadon hagyja viszont a pályatervezés, térkép illesztés, illetve a térképek összefűzésének témakörét, mivel ezeket sokféleképpen lehet megközelíteni és megvalósítani. Az általam felhasznált, esetenként saját módszereket mutatom be a következő pontokban.

A teljes algoritmus egyszeri lefutása 4 nagy részre bontható:

1. Lokális mérésből a térkép részlet előállítása
2. A térképrészlet a korábbi térképhez történő illesztése
3. A két térkép részlet összefűzése (merge)
4. A következő mérési pont meghatározása (maga a NBV)

A negyedik pont magába foglal még egy szemantikailag különálló lépést, a pályatervezést.Az algoritmus az alábbi, konfigurálható értékeket használja:

- Minimális és maximális hatótáv (a távolságszenzoré)
- A távolságszenzorra vonatkozó szög korlát (incidence constaint)
- Robot átmérő
- Maximális távolság két szilárd éllel összekötött pont között
- Térkép összehúzásának mértéke
- Egy egyenes vizsgálatához használt hiba medián maximum
- Generált szabad élek részletessége

4.3.1 Lokális mérés feldolgozása

A NBV algoritmus leírásában a szabad élek hozzáadásán kívül nem tárgyalják a lokális mérés feldolgozását, így ezt én implementáltam, a saját adatstruktúráimhoz, illetve a LIDAR hiányához illeszkedően. Ebben a lépésben a cél, hogy a beérkező méréseket felhasználva előállítsunk egy térkép részletet. Az egyszerűség kedvéért csak az orientáció és távolság méréseket használom, a pozíciót nem, azt a következő lépésben korrigálom. Eredményül tehát egy origo középpontú térkép részlet áll elő. A „mérések sorrendjét" olyan értelemben használom, ahogy a robot az óramutató járásával ellentétes irányú körbefordulása során érzékelte őket.

A mérés feldolgozása 4 lépésből áll:

1. Egybefüggő klaszterekre bontás
2. Egybefüggő mérések szétbontása a kényszerek alapján
3. Pontok számának csökkentése
4. Hézagok feltöltése szabad élekkel

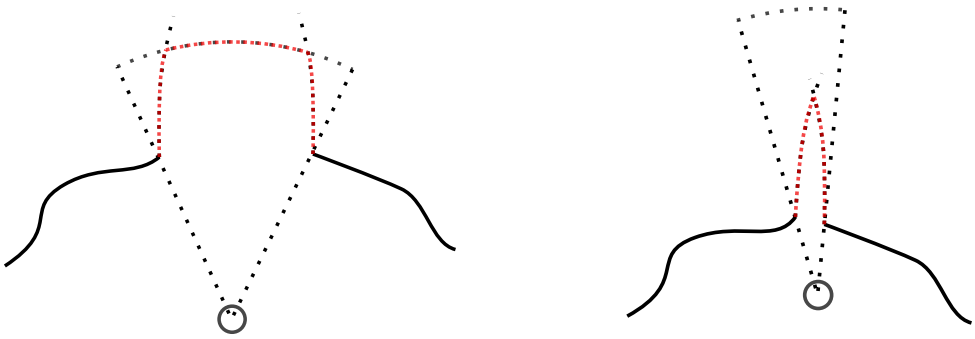
Először az algoritmus eltávolítja a minimális hatótáv alatti pontokat – ilyet csak hibás mérés esetén kaphatunk. Majd az alapján bontja klaszterekre a mérési eredményeket, hogy hol mért hatótávon kívül. Egy klaszterbe azon mérések egymásutánját teszi, amit nem szakít meg tartományon kívüli mérés. Ilyen esetben új klasztert kezd. Ehhez először egy kezdőpontot keres, az első olyan mérést, ami tartományon belül van, de az előtte lévő kívül. Ehhez, ha az első pont tartományon belül van, akkor az utolsó ponttól visszafele, egyébként előrefele indul. Ezután modulo indexelés segítségével körbemegy a pontokon és tartományon belüli értékeket hozzáadja a jelenleg klaszterhez, tartományon kívüli pont esetén tovább lép és az első ismét tartományon belüli ponttól új klasztert kezd. Amennyiben a kezdőpont keresése sikertelen (visszaért a mérés elejére), az azt jelenti, hogy minden mérés tartományon belül, vagy kívül volt. Első esetben az összes pontot tartalmazó, utóbbiban üres klasztert továbbít.

A következő lépésben az előállított klasztereken halad végig az algoritmus, azokat vonal csoportokba szedve. A csoportok csak azonos klaszterben lehetnek, tehát a klaszter végénél mindig új csoportot kezd. Ezen kívül akkor kezd új csoportot, ha két egymásutáni pont távolsága nagyobb a megadott határnál. A két csoportosítás különválasztására azért van szükség, mert a klaszterek közé a szenzorból adódó korlátok alapján kell szabad

éleket kell generálni („tudjuk, hogy ott nincs semmi”), a klaszteren belüli csoportok közé, viszont, ha túl távol van két pont, akkor csak egyenessel tölthetjük ki (nem tudjuk mi van ott). Ezt a korlátot (jelen alkalmazásban) szerintem érdemes a robot szélességére választani, mivel ennél szűkebb helyre úgysem tud bemenni.

A harmadik (opcionális) lépés a mérési pontok számának csökkentése. Ezt érdemes elvégezni, mert az NBV algoritmus sok geometriai eljárást felhasznál, melyeket tipikusan a térkép poligont alkotó minden élre le kell futtatni. Így, ha kevesebb éle van a poligonnak, azzal futásidő spórolható. Az eljárásnak egy töröttvonalat (akár zártat) kell átadni, és egy érdemben hasonló, csak kevesebb pontból álló töröttvonalak készít belőle (például, ha robot egy egyenes doboz oldalát sokszor megmérte, nincs értelme az egyenes mentén/ahhoz közel sok pontot tárolni). Az algoritmus először csoportokra bontja pontokat aszerint, hogy mik lehetnek egy egyenesen. Ehhez a kezdőpontból egyre hosszabb pont sorozatokat tekint. Azt vizsgálja, hogy a köztes pontok távolságának mediánja a kezdő és végpontokat összekötő szakaszhoz képest meghalad-e egy értéket. Addig adja hozzá a csoporthoz a pontokat, amíg a medián határértéken belül van, ha kívül új csoportot kezd. Ezután minden csoportra a legkisebb négyzetek módszerével (Least Square – LS) egy egyenest illeszt. Az így előálló egyenesek metszéspontja fogják a csökkentett számú pontból álló töröttvonal pontjait adni. A kezdő és végpont esetében az eredeti pontokhoz tartózó irány mentén metszi el az egyeneseket. Ha ezek kellően közel vannak, akkor zárt görbeként kezeli és a becsült egyenesek közül az első és utolsó metszéspontja lesz a kezdő és végpontja a szakasznak.

Végül a [1]-ben leírtak szerint, a kapott vonalak közötti réseket szabad élekkel tölti fel. Ehhez először a korábbi szakasz végpontjából és a következő szakasz kezdőpontjából egy-egy exponenciális görbét indítok, az algoritmus leírásában szereplő módon. Ha ezek az ívek a szenzor hatósugarán belül metszik egymást, akkor egy töröttvonallal közelítem őket, mely a korábbi vonal végpontjából indul, érinti a metszéspontot és a következő vonal kezdőpontjába érkezik. Ha a hatósugáron kívül metszik egymást, akkor egy körívet generálok a korábbi vonal végpontja és a következő vonal kezdőpontja közé eső szögtartományon, a szenzor méréshatárának sugarával, majd ezt elmetszem a két exponenciális görbével, és az így kapott alakzatot közelítem a korábbihoz hasonlóan töröttvonallal (14 ábra).



14 ábra Ha az exponenciális ívek a szenzor méréstartományán túl metszik egymást, akkor a szenzor maximális mérési távolságával megegyező sugarú körívvel metszem el őket (balra),

4.3.2 Térképek illesztése

A térképek illesztéséhez létezik több, jellemző illesztés (feature matching) alapú módszer [13] a gépi látásban alkalmazott ORB, SURF, SIFT leírók mintájára. Ezekkel a robot pozícióját is lehetne becsülni, az illesztés alapján. Ezekkel meghatározható a két térkép közötti eltolás és forgatás. Az én alkalmazásomban, viszont a kamera és IMU alapján a robot pozícióját jól ismertnek feltételezem, így külön algoritmust nem használok hozzá, a mérés pozíciójából számolom ki a pontok helyét, így nem szükséges további illesztés.

4.3.3 Térképek összefűzése

A megfelelően eltolt térképek egyesítésével kell előállítani az új méréssel kiegészített térképet. Ez nem egyszerűsíthető két sokszög uniójának képzésre, mivel így

a mérési zaj folyamatosan kijjebb tolná a falakat, valamint az élek szemantikájának megtartása is nehézkes lenne. A feladat elvégzésére saját eljárást dolgoztam ki.

A térképek összefűzéséhez első ötletként egy pusztán lokális információkat felhasználó algoritmust találtam ki, amely végigsétál a térképek peremén és a metszeteknél lokális információk (metsző élek állása, típusaik) alapján dönti el, hogy melyik irányba lép tovább. Ezt az egyszerű megközelítést azért választottam, mert a térkép tárolásához a pontokat sorrendben kell tárolni. Kutatásom alapján csak a csúcspontokból, a sorrend ismerete nélkül visszaállítani egy sokszöget egy bonyolult feladat [14] [15], és sokszor több megoldás is szóba jöhet. Ezzel a megközelítéssel a körüljárási irány nem veszik el.

Ehhez először keres egy olyan pontot az új térképen, ami szilárd élen van és a korábbi térképeken belül, vagy szabad élen van és a korábbi térképeken kívül. Mindkettő biztosan része lesz az összefűzött térképnek, mivel a szilárd éleket nem szeretném kijjebb tolni (inkább legyen korlátozott a biztonságos régió és az tényleg biztonságos, mint fordítva), az utóbbi pedig újonnan felfedezett terület részlet (vagy olyan, ami már fel volt fedezve, de a térkép egy másik részén). Ezután az algoritmus elindul körbe, és a kereszteződéseknél az alábbi módon választ irányt, mindig úgy, hogy a mérési sorrend szerint haladjon körbe:

- Ha mindkét él szilárd: azt követi, amelyik az aktuálisan követett térképen belül van
- Ha mindkét él szabad: azt követi amelyik a követett térképen kívül van
- Ha vegyes: a szilárd élet követi

Amennyiben a követendő pont a másik térképen van, úgy az aktuálisan követett és a másik térképet felcseréli. Ezt addig folytatja amíg körbe nem ér.

Sajnos előfordul, hogy az algoritmus nem az elvárt módon fűzi össze a két térképet, hanem csökkentett területet ad vissza. Erre az esetre implementáltam egy védelmet, mely figyeli, hogy olyan pontot értünk-e el, amit már bejárt az algoritmus. Ha igen, és az nem a kezdőpont, akkor megszakítja az illesztést és visszaadja az eredeti térképet, ezzel új mérést kérve, mivel az összefűzés sikertelen volt.

4.3.4 NBV implementáció

Magát a következő mérési pontot a [1]-ben leírt módon implementáltam. Véletlen pontokat generálok a szabad élek mentén, majd a térképen belül, ezek láthatósági tartományában. Ezután minden ilyen pontba megtervezem az útvonalat. A minden ponthoz elvégzem a potenciálisan felfedezhető terület becslését, a pontból indított sugarakkal. A publikációban [1] megfogalmazott célfüggvényt kiszámolom az elérhető pontokra és ezek közül a legtöbb pontot szerző lesz a következő mérési pont.

Az ehhez szükséges tartalmazás, metszés vizsgálathoz saját algoritmusokat implementáltam, mivel a MATLAB beépítettje rendkívül lassúnak bizonyultak.

4.3.5 Pályatervezés

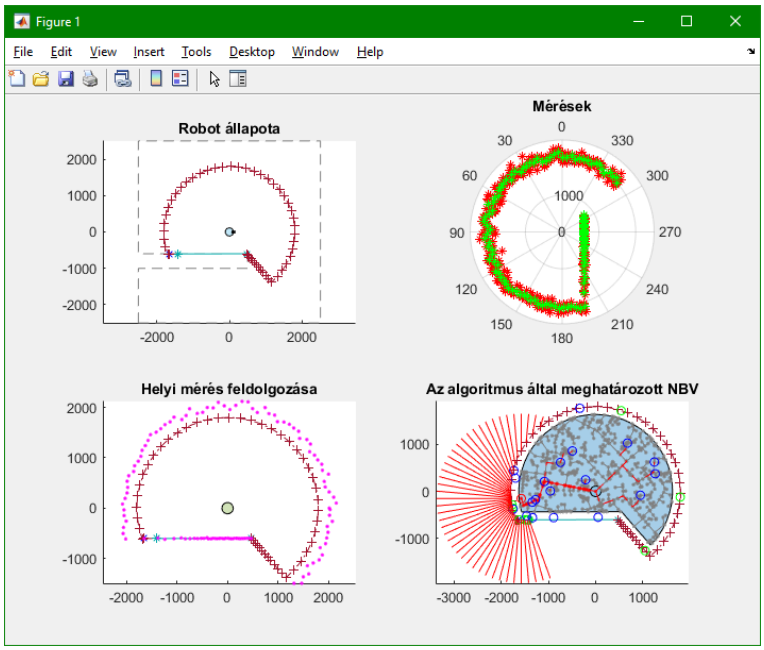
A pályatervező algoritmusok tipikusan pontszerű testeket feltételeznek, a valóságban viszont ez koránt sincs így. A probléma kiküszöböléséhez a térképet összeszűkítem a robot befoglaló körének sugarával (egy kevés biztonsági ráhagyással), ez lesz a szabad régió. Fontos, hogy ez nem skálázás transzformáció, hanem offset (vagy mitered offset), melynek során a szögfelezők mentén kezdjük el a sarokpontokat (és ezekkel együtt az éleket) befelé húzni. Mivel a sokszög a művelet során több komponensre eshet szét, ha ez megtörténik, csak azt választom ki, amelyik tartalmazza a robotot, ez az elérhető régió.

A pályatervezéshez az Rapidly exploring Random Tree (RRT) algoritmust [16] implementáltam. Ez véletlenszerűen választ pontokat az elérhető régióból, a célpontokat is visszahelyettesítve. Ezekhez a pontokhoz próbál útvonalat találni úgy, hogy az aktuális pontot megpróbálja egy egyenessel az addig épített gráfhoz csatlakoztatni. A visszahelyettesítés miatt ez viszonylag gyorsan konvergál egyszerű környezeteknél a célpont(ok)hoz.

Az így megtalált útvonalak esetenként igen kanyargósak és sok pontból állóak lehetnek, messze az időoptimális megoldástól. A roboton további mozgástervezést nem végzek, a korábban leírt P szabályzó biztosítja, hogy a robot elérje a kívánt pozíciót. Az így előállt pálya a sűrű pontok miatt így kis beavatkozó jeleket és lassú működést eredményezett. Ezért itt is egy pont szám redukciót alkalmazok, olyan módon, hogy amíg a kezdőpontból az aktuális pontba létezik a elérhető régión belül haladó szakasz, addig a köztes pontokat kitörlöm. Amennyiben nem, úgy újraindítom az eljárást az aktuális pontból, egészen addig, amíg el nem értem az eredeti szakasz végpontját.

4.4 Megjelenítés

Annak érdekében, hogy könnyen nyomon lehessen követni a robot belső állapotát és a környezetről alkotott képét, készítettem egy egyszerű grafikus megjelenítő felületet is, melyet a 15 ábra mutat be. Ezen bal felül megtekinthető a robot aktuális állapota (pozíció, orientáció), a térkép jelenlegi állapota (folytonos vonal: szilárd élek, szagattott *-gal: szabad élek), illetve alapként (ground truth -szaggatott szürke) a környezet pontos



15 ábra A robot és a térkép aktuális állapotait, méréseket megjelenítő grafikus felület. Balra fönt a robot aktuális állapota, mellette a legutóbbi mérési eredmények (piros: minták, zöld: átlagok), alatta a legutóbbi lokális térkép (rózsaszínnel a mért pontok),

határa. Mellette a nyers mérési eredmények láthatók, pirossal az egyes mintavételezések, zölddel ezek átlaga. Baloldalt alul a legutóbbi mérésből előállított lokális térkép látható, a mért pontokkal (rózsaszín pöttyök). Végül jobboldalt alul a NBV algoritmus eredménye, az elérhető régióval (kék sokszög), a RRT algoritmus által épített fával (szürke), ebből kiemelve az útvonalak a célokhoz (folytonos piros a NBV-hoz, szaggatott a többihez)

4.5 Mérés vezérlés

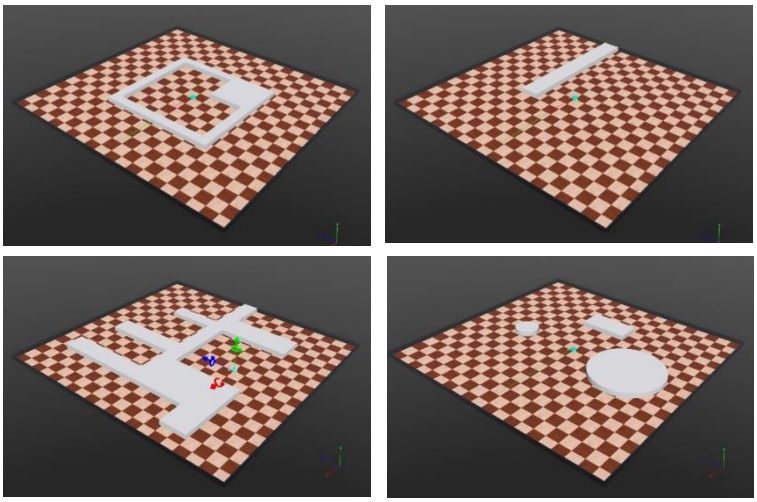
A robot működtetéséhez egy állapotgépes megvalósítást készítettem. Ez egy inicializáló állapotból indul, amelyből rögtön a mérés állapotba jut tovább. Ez egy számlálót tart karban, amely nyilvántartja, hogy hányadik mintavételezésnél járunk. Itt próbálkoztam a törött vonal redukciós algoritmus alapján adaptív lépésköz választással,

de ez nem hozott javulást. A robot minden pontban 5 mérést végez, ezek átlagolásával kellően csökkenthető a mintavétel zaja, hogy azt eredményesen tudjuk feltérképezésre használni.

Ha a robot körbefordult, és végzett a mérésekkel, lefuttatja a feltérképező algoritmust, amely megadja az új pontba vezető út pontjait. Ennek a követése a következő állapotban történek. Egyenként állítom be alapjelnek a pontokat, akkor lépve a következőre, ha a robot egy adott értéknél közelebb van a jelenlegi célhoz. Miután a robot elérte az út végpontját, visszalép a mérés állapotba, a számláló visszaállításával és elölről kezdődik a lefutás.

4.6 Teszt eredmények

Az algoritmus tesztelésére 4, egyre nehezedő környezetet készítettem, melyeket a 16 ábra jelenít meg. A lokális mérési adatok feldolgozásának teszteléséhez először egy mérésből felfedezhetőt, ezután egy egyszerűbb több lépésből felfedezhetőt, majd egy több részlettel, bonyolultabb felépítésűt, végül pedig egy szegetekből állót, amelyet jelenleg nem elvárt, hogy kezeljen, de a viselkedése tanulmányozható volt.

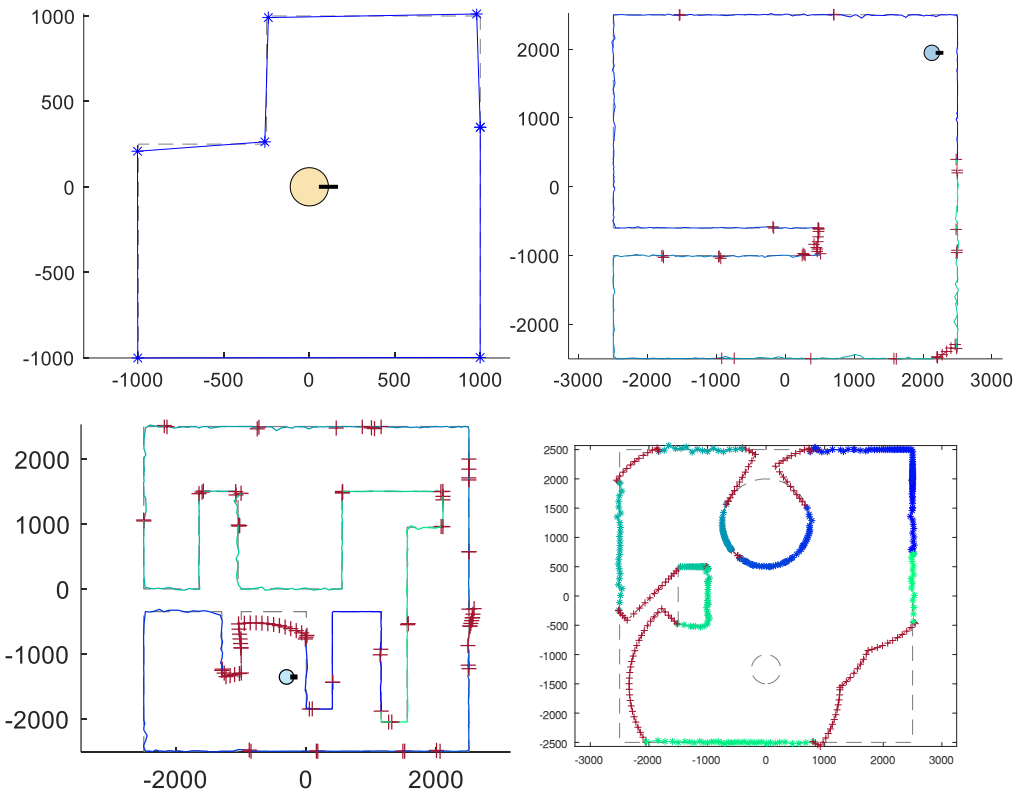


16 ábra A kipróbált környezetek. Elsőnek egy helyből is feltérképezhető környezetet használtam, majd egy egyszerű, de mozgást igénylőt. Ezután egy összetettebb környezetre is teszteltem az algoritmust, majd sziget jellegű akadályokkal is megvizsgáltam a működés

A tesztpályákat a robot képes volt feltérképezni a komplikáltabb, egymáshoz közeli falakat tartalmazó környezeteket is, a zajos mérések ellenére is. Sikeresen tudott végig navigálni rajtuk, a szűkebb részeken is, ütközésmentesen. A hatékonyságával viszont nem vagyok megelégedve. Túl sokszor fordul elő, hogy az algoritmus nem tudja

sikeresen összefűzni a térképeket és új mérést kér. Ezzel jóval hosszabbá téve a feltérképezést.

Sziget akadály esetén is működött egy bizonyos fokig az algoritmus. Az esetek nagy részében egyszerűen nem tudta leválasztani a szigeteket, amely jelentős problémát nem okoz (nem fog ütközni a robot), csak nem használja ki az átjárókon keresztüli rövidebb utat. Néhány esetben viszont előfordult, hogy az összefűző algoritmus teljesen kihagyta a szigetet, azt is a biztonságos régióhoz adva. Az eredményeket a 17 ábra mutatja be. A teszt során megfigyeltem, hogy az összefűző algoritmus dolga könnyebb, ha nem redukálom a törött vonalak pontszámait, így azt átmenetileg kivettem a lépések közül, az összefűzést igénylő teszteseteknél. Ennek az eredménye az azoknál látható recésebb térkép szél.



17 ábra A tesz környezetek eredményei. Látható, hogy az algoritmus működi, sikeresen feltérképezte a környezeteket. A jobb alsó, szigetes esetnél előfordul, hogy az összefőzés kihagyja a szigetet, így egy akadály "eltűnik a térképről"

5 Implementáció valós környezetben

A fejlesztés során végig a végcél az algoritmus valós környezetben történő kipróbálása volt. Ehhez a már meglévő implementációt egyéb elemekkel kellett kiegészíteni, hogy a robot számára elérhetőek legyenek az eredményes navigációhoz szükséges információk. Ebben a fejezetben az így előállt környezetet tekintem át, illetve az elért eredményekre is kitérek.

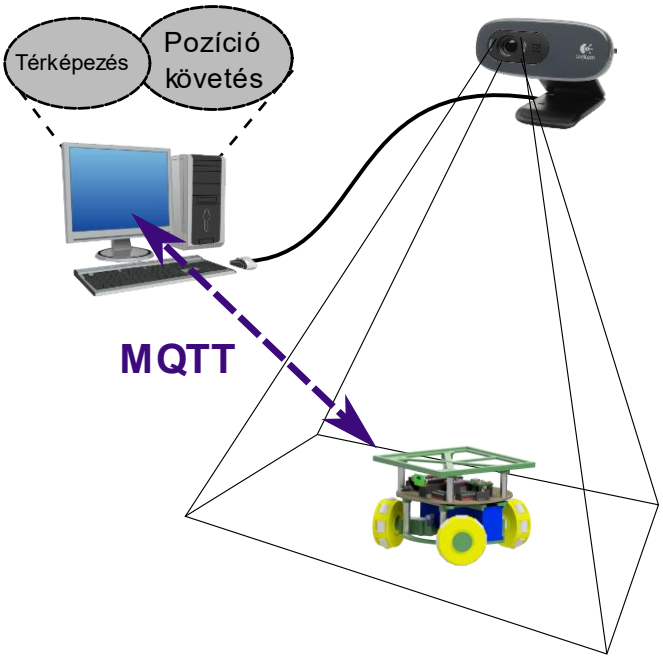
5.1 Teszt összeállítás ismertetése

Az implementált algoritmus számításigényesnek bizonyult a sok véletlen szám generálás miatt. Így nyilvánvaló volt, hogy ezt a mikrokontrolleren nem tudom megvalósítani. Ezért egy kliens alkalmazást is létre kellett hozni, ami lebonyolítja a kommunikációt a robottal és megtervezi számára a pályát. Ezzel a megoldással a későbbiekben egyszerűn becsatolható több robot is a rendszerbe.

További megoldandó feladat volt, hogy a robot pozíció és orientáció visszacsatolást kaphasson. A fedélzeti inerciális szenzorból is kinyerhetőek ezek az adatok, viszont a pozícióbecsléshez kétszeres integrálást kell alkalmazni, melynek hatására a becsült pozíció hamar pontatlanná válik (drift jelenség). A feladat megoldható lett volna többek között hiperbolikus navigáció segítségével, illetve valamilyen gépi látás alapú módszerrel. Az előbbi megvalósítása a tanszéki tapasztalatok alapján rendkívül körülményes, így a választás a kamera alapú megoldásra esett. ezen Egy kamerát helyeztem el a környezetben, amely felülről látja a robotot. A robot tetejére pedig egy markert helyeztem, a könnyű felismerhetőség érdekében.

A különböző komponensek közötti kommunikációra az MQTT protokollt választottam. Ezen keresztül az egyes komponensek topic-okra iratkozhatnak fel, illetve ezekre publikálhatnak üzeneteket. Az üzenetek megfelelő klienshez történő továbbítását a központi bróker végzi. Ennek a protokollnak a segítségével könnyen megvalósítható a platformfüggetlen kommunikáció, és az üzenetek topic-okba szervezésével jól átlátható kommunikációs séma alakítható ki. Ezen felül egy újabb adó vagy vevő elhelyezése a rendszerbe nem igényli az addigiak módosítását, ezáltal könnyen kiterjeszthető több ágenses rendszerre.

A fejlesztés megkönnyítéséhez az üzenetek felépítését az általam korábban fejlesztett diagnosztika alkalmazáshoz igazodva határoztam meg, hogy azon követhessem az algoritmus futásához nem szükséges jeleket. Az összeállítást a 18 ábra szemlélteti.



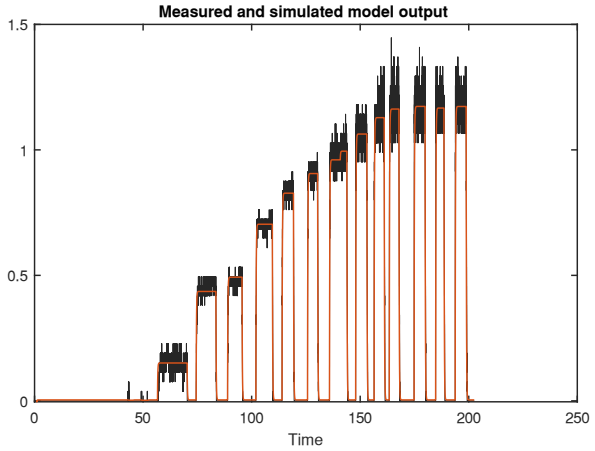
18 ábra A fizikai környezetben működő rendszer komponensei. A robot vezeték nélkül kommunikál a klinessel, MQTT protokoll segítségével. A kamera kiszervezhető lenne egy egykártyás számítógépre (pl.: Raspberry PI), így nincs a kábel hosszához kötve az elrendezés

5.2 Egyes komponensek megvalósítása

5.2.1 Beágyazott szoftver

A beágyazott szoftver elkészítéséhez a FreeRTOS operációs rendszert használtam, valamint az Arduino környezetet, mivel ehhez sok szenzorkezelő és egyéb könyvtár elérhető. A feladat sokrétűsége miatt (kommunikáció, szabályzók, mérés) a többszálú megvalósítást tartottam célszerűnek. Mivel az ESP32 mikrokontrollerben két mag áll rendelkezésre, az egyiket a kemény valósidejű (hard real-time), szabályzást végző taszkoknak allokáltam, a másikat pedig a kevésbé kritikus időzítésű kommunikáció és feltérképezést kezelő taszkok számára.

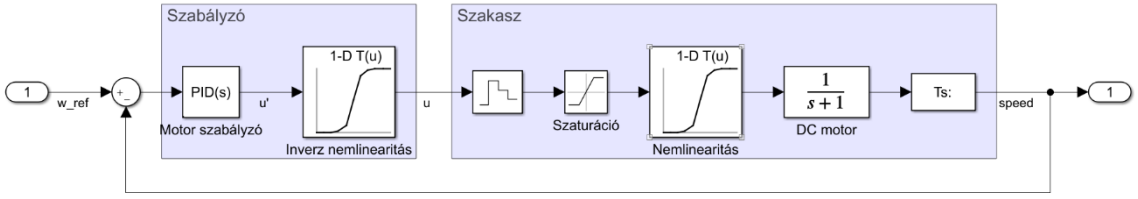
A robot irányításához a szimulációban is használt kétszintes szabályzó struktúrát alkalmaztam, ezeket két külön szálon futtatva. Itt viszont már nem lehetett elnagyoltan kezelni a kerekek forgási sebességét. Így MATLAB segítségével identifikáltam a motorokat Hammerstein-Wiener modellt és bemeneti nemlinearitást feltételezve. Az identifikált rendszer kimenete 90%-ban egyezik meg a mérési adatokkal, ez elégségesnek tekinthető melyet a 19 ábra is szemléltet.



19 ábra A DC motor identifikációja. Az identifikált modell kimenete 90%-os egyezést mutatott a mérési adatokkal

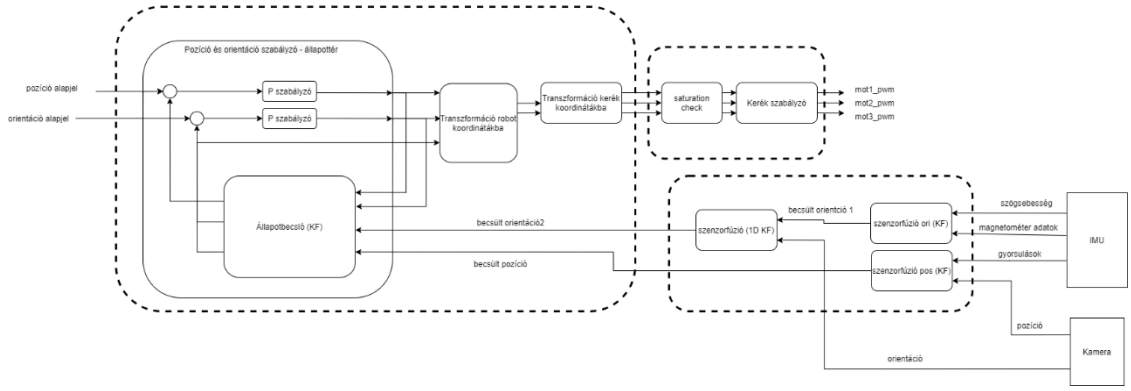
Az így kapott lineáris modellhez PID szabályzót terveztem a MATLAB erre szolgáló eszközével. Itt a gyors beállással szemben a robosztus működést tartottam szem előtt, továbbá ellenőriztem, hogy a szabályzó által kért beavatkozó jel ne haladja meg a motor képességeit. Az így előállt diszkrét idejű szabályzó impulzusválasza már könnyen implementálható a mikrokontrolleren. A bemeneti nemlinearitásokatat az inverz lookup table segítségével, lineáris interpoláció megvalósításával kezeltem. Az elintegrálódás elkerülésére, a robot kinematikájánál leírt telítődést is figyelembe vettem (20 ábra).

Állapotbecslés



20 ábra A kerékszabályzó hatásvázlata. A pontosabb szabályzás érdekében a motor statikus nemlinearitását is kezeltem

A kerékszabályzásra épülő pozíciószabályzást a szimulációhoz képest állapotbecslés megvalósításával egészítettem ki. Mivel a robot állapota lineáris rendszernek tekinthető (az omnidirekcionális megvalósítás miatt), a becslésre Kálmán-szűrőket alkalmaztam. Annak érdekében, hogy minden információt felhasználjak a becsléshez – ezáltal jobb eredményt várva – szenzorfúzió megvalósítására is felhasználtam az eljárást [17].



21 ábra A robot teljes szabályzási és állapotbecslési struktúrája

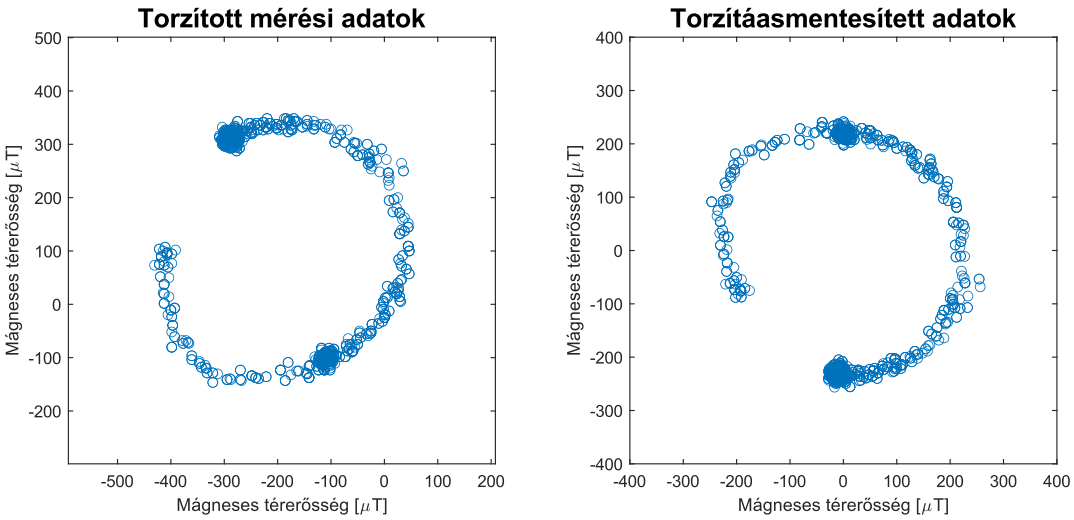
Először a giroszkópból és magnetométerből érkező adatokat fuzionálom az orientáció becsléséhez, majd az így előállított becsléshez veszem hozzá a kamerából érkező értéket. Ezzel párhuzamosan elvégzem a pozíció adatok fuzionálását a gyorsulásmérő és kamera adatok felhasználásával. Az így előállított becslés lesz az állapotbecslő mérési bemenete. Ezt kombinálva a rendszer elméleti működéséről alkotott képünkkel állítom elő a végső állapotbecslést, amely alapján a pozíciószabályzás történik. Ehhez már csak egyszerű P tagokat alkalmazok, mivel az integrátor hatás a kerék sebesség szabályzójában már benne van (21 ábra). A Kálmán-szűrők inicializálásához szükséges kovariancia mátrixokat a robot nyugalmi állapotában történt mérések segítségével határoztam meg, az egyes zajokat függetlennek feltételezve.

Először a giroszkópból és magnetométerből érkező adatokat fuzionálom az orientáció becsléséhez, majd az így előállított becsléshez veszem hozzá a kamerából érkező értéket. Ezzel párhuzamosan elvégzem a pozíció adatok fuzionálását a gyorsulásmérő és kamera adatok felhasználásával. Az így előállított becslés lesz az állapotbecslő mérési bemenete. Ezt kombinálva a rendszer elméleti működéséről alkotott képünkkel állítom elő a végső állapotbecslést, amely alapján a pozíciószabályzás történik. Ehhez már csak egyszerű P tagokat alkalmazok, mivel az integrátor hatás a kerék sebesség szabályzójában már benne van. A Kálmán-szűrők inicializálásához szükséges

kovariancia mátrixokat a robot nyugalmi állapotában történt mérések segítségével határoztam meg, az egyes zajokat függetlennek feltételezve.

IMU kalibráció

Annak érdekében, hogy az IMU egység mérései felhasználhatóak legyenek, szükség van bizonyos hatások ellensúlyozására. A gyorsulásmérő és a giroszkóp esetében ezek egy bias értéket jelentenek, melyre nyugalmi állapotban történt mérések átlagolásával jó becslés adható. Magnetométer esetében viszont a kemény és lágy mágneses hatások rendre további offszet és torzulásos zavarást eredményeznek [18]. Ezek kompenzálásához a robotot körbe forgatva gyűjtöttem adatot és végeztem el a kompenzálást. Mivel a robot csak síkban mozog, ezt elég volt 2 dimenzióban, egy tengely körül megtennem. A kalibráció eredményét a 22 ábra mutatja.

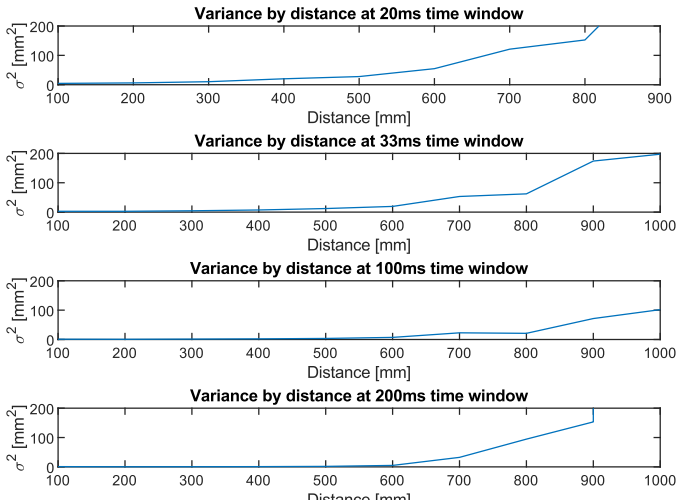


22 ábra A magnetométer mérési pontjai kalibráció előtt és után

Távolságszenzor kezelése

A távolságmérő szenzorok mérési bizonytalansága általában függ a megmérni kívánt távolságtól. Emellett a szenzor mérési ablaka konfigurálható. Annak érdekében, hogy kipróbáljam milyen beállítással érdemes használni és milyen távolságkorlátok mellett (meddig tekinthető „pontosnak” a mérés) megvizsgáltam, hogy különböző távolságokon (10-100cm, 10cm lépésközzel) az egyes időablakok esetén (20ms, 33ms, 100ms, 200ms) milyen eredményt kapok (23 ábra). A mérések alapján megfigyelhető, hogy a variancia (a vizsgált értékek közül) 100ms időablak esetén a legkedvezőbb. Itt 60cm alatt rendkívül kicsi a bizonytalanság és utána 80cm-ig is alacsony marad, csak

ezután kezd el jelentősen emelkedni, ellenben a többi ablakkal, ahol már jóval hamarabb megfigyelhető a variancia megugrása. Így 100ms időablakot és 80cm-es mérési korlátot alkalmaztam.



23 ábra A mérések varianciája különböző mérési időablakok esetén, 10-100cm méréstartományon

Állapotgép

A robot magasz szintű működésének irányításához itt is a szimulációban használt állapotgépet alkalmaztam, annyi módosítással, hogy a pályatervezés állapot helyett útvonalra vár állapotba kerül a robot a mérés elküldése után, egészen a válasz beérkezéséig. Mivel a pozíciószabályzás enyhén mindig jitterel, és azt tapasztaltam, hogy a robot jól meg tud egyhelyben fordulni, ezért a körbefordulás idejére a pozíció beavatkozást letiltom.

5.2.2 Pozíció becslés kamera segítségével

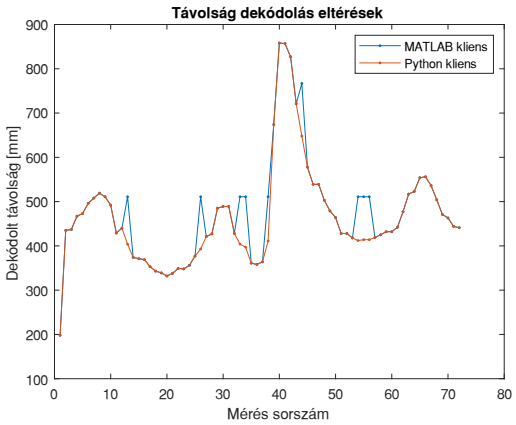
Ahogy korábban írtam, a robot pozíció és orientáció becslésének pontosításához egy felülről a környezetre néző kamerát alkalmaztam és a robot tetejére egy markert helyeztem. Az elérhető sokféle marker típus közül az ArUco markerekre esett a választásom. Legfőbb előnye, hogy az OpenCV könyvtár beépítetten támogatja, és képes a marker pozícióját és orientációját megbecsülni. Ezen kívül a marker egy azonosító számot kódol, amely lehetővé teszi, hogy több ágenses rendszer esetén könnyen azonosítsuk, melyik robothoz tartozik a marker. A felismerést tovább könnyíti, hogy generálhatunk saját marker könyvtárat, olyan elemszámmal amire szükségünk van és a rendszer biztosítja a lehető legnagyobb különbséget a markerek között, így a számunkra

szükséges mennyiséghez a lehető legjobban megkülönböztethető markereket tudunk generálni.

A kamera felszerelése közben hamar rájöttem, hogy a pontosan függőleges optikai tengely beállítása nehéz feladat lenne, így felmerülhet olyan probléma, miszerint a képsík nem párhuzamos a robot mozgási síkjával, így pontatlan lenne a pozícióbecslés, ráadásul változó mértékben. Ennek elkerülése érdekében a felismerés eredményéül előálló pozíció vektort a marker X és Y tengelye által meghatározott síkba vetítem, és origónak a kamera optikai tengelyének ugyanerre a síkra vett döféspontját tekintem. Ezzel a marker pozícióját mindig a saját síkjában kapom meg, így elkerülhetők a korábban említett hatások. A kamera kalibrációját a MATLAB erre szolgáló segédeszközével végeztem el, a szokásos sakktábla jellegű mintát alkalmazva.

5.2.3 Kliens alkalmazás

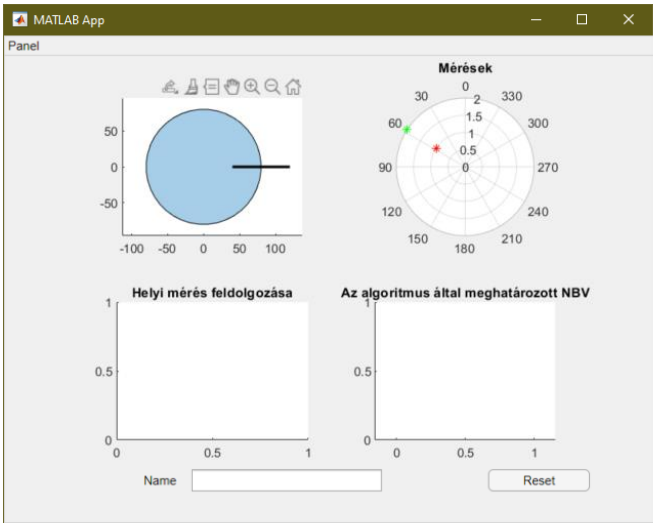
A szimuláció során a MATLAB minden funkciója közvetlenül elérhető volt, ugyanez viszont a valós környezet esetén nem áll fent. A feltérképezés elvégzéséhez és az adatok megjelenítéséhez ezért készítettem egy MATLAB App alkalmazást. Ez a feltérképező algoritmuson kívül egy MQTT klienst tartalmaz, amely a beérkezett mérés után meghívja a feltérképező eljárást és a robotnak visszaküldi a megtervezett útvonalat. Az alkalmazásba átemeltem a korábban elkészített grafikus megjelenítési elemeket is, hogy követhető legyen a feltérképezés. Itt problémát okozott, hogy az MQTT MATLAB implementációja nem kezeli jól a bináris üzeneteket. A könyvtár szöveges üzeneteket vár, és a feldolgozás valamely lépésekor emiatt egyes bájtok értéke megváltozik (24 ábra). Sajnos nem sikerült megoldást találni a problémára, így a mikrokontrollernek muszáj



24 ábra A MATLAB által dekódolt értékek nem egyeztek meg a Python által dekódolt (helyes) értékekkel, így a bináris helyett a pazarlóbb, JSON formátumú adatküldést kellett alkalmaznom

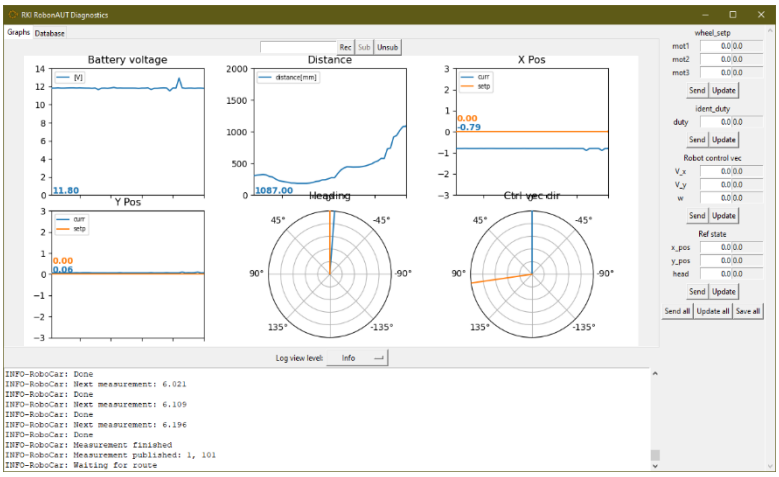
JSON formába kódolnia a mérési eredményeket, amely további erőforrást igényel és sokkal pazarlóbb, viszont szerencsére erre maradt még kapacitása.

Az alkalmazásba becsomagoltam a korábban említett grafikus kijelzőket, valamint hozzáadtam egy Reset gombot, amivel újraindítás nélkül törölhető az aktuális térkép állapot, illetve egy beviteli mezőt, amiben megadhatjuk a mérés elnevezését. A beérkezett regisztrátumokat az alkalmazás egy ilyen nevű mappába menti el, hogy később rekonstruálható legyen a mérés. Az alkalmazás felületét a 25 ábra szemlélteti.



25 ábra A valós rendszerhez készített MATLAB alkalmazás felülete. Megtalálhatok rajta ugyanazok a kijelzők, mint a szimulációhoz készített felületen, valamint egy visszaállító gomb és egy szövegmező, ahol elnevezhetjük a mérést

A fejlesztést könnyítésére a robotba implementáltam telemetria adatok küldését is, melyet a más projektjeimhez is használt, könnyen konfigurálható diagnosztika alkalmazásom továbbfejlesztésével tudtam megjeleníteni. Ezt nem az algoritmus kliens



26 ábra A felhasznált saját fejlesztésű diagnosztika felület

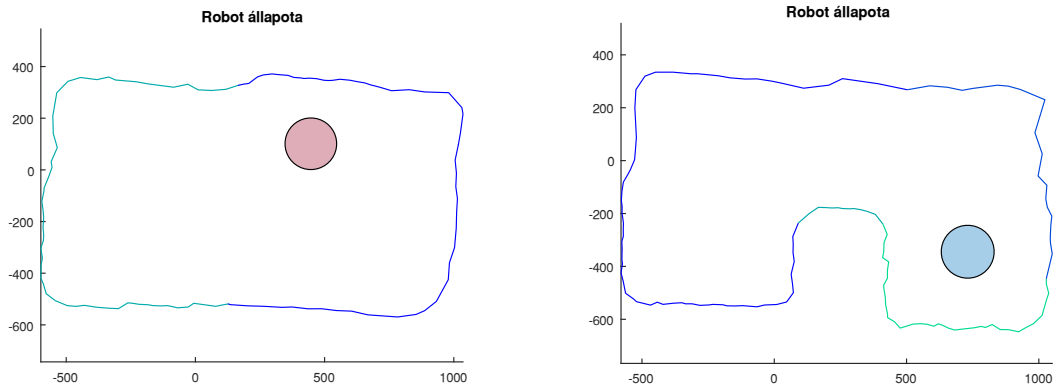
alkalmazásába implementáltam, mivel nem kötődnek a feltérképezéshez, debug információkat jelenít meg (26 ábra).

5.3 Teszt eredmények

A valós környezetben hely, illetve eszköz hiányában nem tudtam annyi féle teszt környezetet összeállítani, mint a szimuláció esetében, viszont ezek is mutatják, hogy valós környezetben is képes az algoritmus a működésre. Először itt is egy egyszerű, téglalap alapú környezetet állítottam össze, hogy az alapvető működésről meggyőződjek, mielőtt tovább lépek. Ezután betettem egy akadályt, ami kitakarta a környezet egy részét, illetve egy szűkebb átjárót hagyott csak a robot számára.

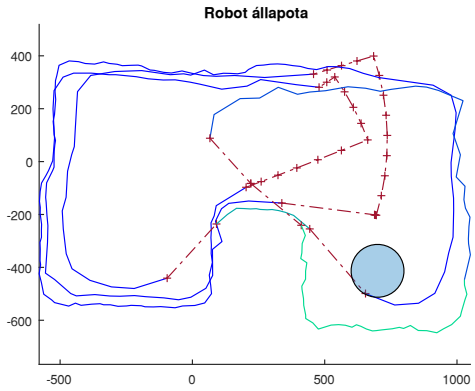
27. ábra A valós teszteléshez használt 140x82cm méretű pálya, valamint a 23x33.5cm méretű akadály

Az első, egyszerűbb teszteset nem okozott problémát a robot számára, két mérésből feltérképezte az elvárásoknak megfelelően. A bonyolultabb környezet viszont már nehézséget okozott az összefűző algoritmus számára. Itt sok köztes, szükségtelen



28. ábra A robot által feltérképezett területek. Látható, hogy vízszintes irányban kicsit hosszabb a pálya és az akadály is, mint kellene lennie

lépés kellett, hogy a térkép összeálljon. A tesztek alapján a robot navigációs és feltérképező algoritmusa működőképes, a feladatot végrehajtotta. A feltérképezendő terület mérete 140x82cm volt, melyen egy 23x33.5cm méretű akadályt helyeztem el a második esetben. Y irányban a fentebb ábráról leolvasva a robot által alkotott térképen is ilyen méretben jelennek meg a pontok, X irányban viszont az ábráról is látszik, hogy valamilyen fajta elcsúszás, vagy skálázási hiba van. Az egyes lokális térképeket egymásra rajzolva észrevehető, hogy azok a vonalak, melyeknek egymást közelítőleg fedniük kellene kellen egymást el vannak csúszva. A távolságszenzor mérései alapján láthattuk, hogy annak pontatlansága ennél jóval kisebb. Ebből arra következtetek, hogy a kamera által szolgáltatott pozícióadat nem annyira pontos, mint azt az első tesztek alapján tűnt, ez további vizsgálatot kíván.



29. ábra Az egymásra rajzolt lokális térképek. Mivel azon élek között, melyeknek egymáson kellene lenniük a távolságmérő szenzor hibájánál nagyobb eltérés van, arra következtethetünk, hogy a pozícionáló rendszer nem kellő pontosságú

6 Tapasztalatok és további fejlesztési irányok

A munkám során megterveztem, elkészítettem, majd a gyakorlatban is teszteltem egy feltérképező algoritmusok fejlesztését segítő rendszert, valamint ehhez egy algoritmust is implementáltam. Mivel a rendszer már működő képes, a továbbiakban a feltérképező algoritmusok finomítására, továbbfejlesztésére és kipróbálásra nagyobb hangsúlyt lehet fektetni.

Ahogyan azt már az algoritmus leírásánál is említettem, előfordul, hogy a két térkép összefűzése nem sikerül. Jelenleg ez az algoritmus gyenge pontja, mivel emiatt a robotnak sokkal több energiába kerül felfedezni a környezetet, mint az szükséges lenne. A probléma megoldására több lehetőség is felmerült bennem, ezeket szeretném megvizsgálni és működőképességüket kipróbálni. Az első ötletem az lenne, hogy az új mérésnek azon pontjaira koncentrálni, amely a már felfedezett területről szabad élen látszik, mivel ez a ténylegesen új terület. Ez viszont kizárná a lehetőségét, hogy a későbbi mérésekkel az addigi térkép is pontosítható legyen. Ennek eléréséhez egy valószínűségi, vagy függvényapproximációs megközelítése is felmerült bennem, mely során az egyes pontokhoz egyfajta magabiztosságot rendelnék (a távolságmérő szenzor zajkarakterisztikájából kiindulva), és azokon a helyeken, ahol az új és a korábbi mérések átlapolnak ezek alapján, a mérések ötvözésével pontosítható lenne a robot környezetről alkotott képe, és ellenállóbb lesz a kiugró mérési hibákkal szemben.

A valós tesztkörnyezetekben további problémát jelentett, a célpont jelöltek generálása. A jelöltek létrehozásánál nem az elérhető régiót, hanem a teljes térképet vettem alapul. Ez a szimulációban nem okozott gondot, mivel a térkép széli, nem elérhető sáv jóval kisebb területű, mint az elérhető. A valóságban viszont kisebb környezetet tudtam csak építeni, emiatt összemérhetővé vált az elérhető és a nem elérhető terület, melynek eredményeképpen sok jelölt esett az elérhető régión kívülre. Ez algoritmikailag egyszerűen javítható, viszont a tesztelés során már nem maradt időm a programban megvalósítani.

Ezen kívül szeretném a hardvert is tovább fejleszteni, hatékonyabbá, könnyebben használhatóvá tenni további visszajelző LED-ek és gombok hozzáadásával, ezzel is könnyítve a fejlesztéseket, valamint felkészíteni a robotcsapatban való működésre. Hosszabb távon az algoritmus fejlesztését is a multiágensű feltérképezés irányába

szeretném elvinni, valamint az összefűző algoritmus fejlesztésével a szigeteket tartalmazó, valamint dinamikus környezetek kezelését is lehetővé tenni.

Irodalomjegyzék

[1] H. H. G.-B. a. J.-C. Latombe, „Navigation Strategies for Exploring Indoor Environments,” *The International Journal of Robotics Research*, pp. 829-848, 1 Október 2002.

[2] S. K. B. Prabin Kumar Panigrahi, „Localization strategies for autonomous mobile robots: A review,” Journal of King Saud University –, Department of Computer Science and Engineering, C. V. Raman Global University, Bidya Nagar, Mahura, Janla, Bhubaneswar, Odisha 752054, India, 2020.

[3] L. P. M. F. A. P. O. R. Sergio Cebollada, „A state-of-the-art review on mobile robotics tasks using artificial intelligence,” Elsevier Ltd., Department of Systems Engineering and Automation, Miguel Hernández University, Elche, 03202, Spain, 2020.

[4] K. T. H. D. Burak Kaleci, „2DLaserNet: A deep learning architecture on 2D laser scans for semantic,” Elsevier B.V., Eskisehir Osmangazi University, Eskisehir, Turkey, 2021.

[5] M. F. a. I. Afanasyev, „Comparison of Various SLAM Systems for Mobile Robot in an Indoor Environment,” in *International Conference on Intelligent Systems (IS)*, Funchal, Portugal, 2018.

[6] Y. C. Heonmoo Kim, „Location estimation of autonomous driving robot and 3D tunnel,” International Journal of Mining Science and Technology, Department of Energy Resources Engineering, Pukyong National University, Busan 48513, Republic of Korea, 2021.

[7] J. O. T. P. S. B. Ana Batinovic, „Decentralized Strategy for Cooperative Multi-Robot Exploration and Mapping,” in *IFAC PapersOnLine*, University of Zagreb, Faculty of Electrical Engineering and Computing,, 2020.

[8] J. R.-S. N. P. J. G.-J. D. Fernandez-Chaves, „ViMantic, a distributed robotic architecture for semantic mapping in,” Elsevier B.V., n, Biomedical Research Institute of Malaga, 2021.

[9] J. C. B. C. C. Krisztián Balázs Kis, „A simultaneous localization and mapping algorithm for sensors with low sampling rate and its application to autonomous mobile robots,” in *Elsevier B.V.*, SZTAKI, Eötvös Loránd Research Network, Kende u. 13-17., Budapest H-1111, Hungary, 2021.

[10] S.-E. Oltean, „Mobile Robot Platform with Arduino Una and Raspberry Pi for Autonomous Navigation,” in *Elsevier B.V.*, University of Medicine, Phamacy, Sciences and Technology of Targu Mures, Nicolae lorga st., No.1. 540088 Targu Mures, Romania, 2019.

[11] H. R. Milad Nazarahari, „40 years of sensor fusion for orientation tracking via magnetic and inertial measurement units: Methods, lessons learned, and future challenges,” Elsevier B.V., Department of Mechanical Engineering, University of Alberta, Canada, 2021.

[12] K. M. L. é. F. C. Park, Modern Robotics: Mechanics, Planning, and Control, Cambridge: Cambridge University Press, 2017.

[13] L. S. O. C. F. Edson Justino, Reconstructing shredded documentsthrough feature matching, Forensic Science International: Elsevier B. V., 2005.

[14] L. D. a. G. Shute, „Polygonizations of Point Sets in the Plane,” Discrete & Computational Geometry, University of Minnesota, Duluth, MN 55812, USA, 1988.

[15] M. D. G. D. S. A. N. Serafino Cicerone, „On the effectiveness of the genetic paradigm for polygonization,” Elsevier B.V., Information Processing Letters 171, 2021.

[16] S. M. LaValle, „Rapidly-Exploring Random Trees: A New Tool for Path Planning,” Department of Computer Science, Iowa State University, Ames, IA 50011 USA, 1998.

[17] E. D. D. P. P. V. Francois Caron, „GPS/IMU data fusion using multisensor Kalmanfiltering: introduction of contextual aspects,” Elsevier B. V., Science Direct - Information Fusion, 2004.

[18] M. K. a. T. B. Schön, „Magnetometer Calibration Using Inertial Sensors,” IEEE, IEEE Sensors Journal, 2016.