


UNIVERSITATEA „SAPIENTIA” DIN CLUJ-NAPOCA
FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE,
TÎRGU MUREȘ
SPECIALIZAREA AUTOMATICĂ ȘI INFORMATICĂ APLICATĂ

Investigarea si implementarea
metodelor SLAM
PROIECT DE DIPLOMĂ

Coordonator științifici:
Prof.dr.ing. Márton Lőrinc
Asistent drd. Fehér Áron

Absolvent:
Krizbai Csaba

2021

UNIVERSITATEA „SAPIENTIA” din CLUJ-NAPOCA Facultatea de Științe Tehnice și Umaniste din Târgu Mureș Specializarea: Automatică și informatică aplicată		Viza facultății:
LUCRARE DE DIPLOMĂ		
Coordonator științific: Prof.dr.Márton Lőrinc asistent drd. Fehér Áron	Candidat: Krizbai Csaba Anul absolvirii: 2021	
<p>a) Tema lucrării de licență: Investigarea si implementarea metodelor SLAM</p> <p>b) Problemele principale tratate: Probleme teoretice: - Localizarea roboților prin metode SLAM - Controlul roboților mobili Probleme practice: - Proiectarea și realizarea unui sistem de localizare folosind robot mobil</p> <p>c) Desene obligatorii: - Schema bloc a sistemului de comandă a robotului - Diagramele programului de control și localizare - Grafice cu rezultate de măsurare</p> <p>d) Softuri obligatorii: - Programul de comandă a robotului - Program pentru vizualizare a rezultatelor localizării</p> <p>e) Bibliografia recomandată: - [1] G.N Desouza, A.C. Kak, Vision for mobile robot navigation: a survey, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 24, 2002 - [40] Peter Corke, Robotics, Vision and Control Fundamental Algorithms in MATLAB, Springer, 2011</p>		
<p>f) Termene obligatorii de consultații: săptămânal g) Locul și durata practicii: Universitatea Sapientia, Facultatea de Științe Tehnice și Umaniste din Târgu Mureș, laborator 243, respectiv on-line Primit tema la data de: 15.05.2020 Termen de predare: 06.07.2021</p>		
Semnătura Director Departament Semnătura responsabilului programului de studiu	Semnătura coordonatorului  Semnătura candidatului 	

Declarație

Subsemnata/ul *Krizbai Csaba*, absolvent(ă) al/a specializării *Automatică și informatică aplicată*, promoția...2020/2021 cunoscând prevederile Legii Educației Naționale 1/2011 și a Codului de etică și deontologie profesională a Universității Sapiientia cu privire la furt intelectual declar pe propria răspundere că prezenta lucrare de licență/proiect de diplomă/disertație se bazează pe activitatea personală, cercetarea/proiectarea este efectuată de mine, informațiile și datele preluate din literatura de specialitate sunt citate în mod corespunzător.

Localitatea, Târgu Mureș

Data: 06.07.2021

Absolvent

Semnătura.....*Krizbai Csaba*.....

Analysis and implementation of SLAM methods

Abstract

One of today's most popular research areas is the environmental sensing of autonomous robots and vehicles. To analyze their environment, we can use light detection and ranging technology (LIDAR), which will record the spatial position of obstacles in their environment in the form of point clouds. These measurements can be used to produce relatively accurate geometric mapping. The popularity of the LIDAR sensor is also due to the fact that its measurements are not affected by external illumination.

The aim of my thesis is to implement a mobile robot that is able to map an unknown environment using a LIDAR sensor and then navigate autonomously in the location it surveys. I solved this task using the SLAM (Simultaneous Localization and Mapping) algorithm. The data processing unit of the robot is a Raspberry Pi single board computer. The mapping is done by a Hokuyo URG type LIDAR sensor, the acceleration and rotation measurement is done by an inertial measurement unit (IMU). The noise values provided by the IMU are attenuated using an Infinite Impulse Response (IIR) filter. The resulting robot orientation value and the data extracted from the LIDAR point cloud were fused using the SLAM algorithm. The measured data and the results obtained from the algorithms are displayed. I implemented a graphical interface responsible for robot configuration and monitoring, written in Python and hosted on Raspberry.

The experiments carried out show that the developed measurement method is well applicable to the navigation of mobile robots in the presence of obstacles.

Keywords: SLAM, LIDAR measurement, mobile robot

Investigarea si implementarea metodelor SLAM

Extras

Unul dintre domeniile de cercetare populare în prezent este detectarea mediului de către roboți și vehicule autonome. Pentru a le analiza mediul înconjurător, putem folosi dispozitive de localizare cu laser (LIDAR), care înregistrează poziția spațială a obstacolelor din mediul lor sub formă de nori de puncte. Aceste măsurători pot fi utilizate pentru a produce o cartografiere geometrică precisă. Popularitatea senzorului LIDAR se datorează, de asemenea, faptului că măsurătorile sale nu sunt afectate de iluminarea externă.

Scopul tezei mele este de a implementa un robot mobil capabil să cartografieze un mediu necunoscut folosind un senzor LIDAR și apoi să navigheze autonom în locația pe care o studiază. Am rezolvat această sarcină folosind algoritmul SLAM (Simultaneous Localization And Mapping). Unitatea de procesare a datelor a robotului este un calculator monoplacă Raspberry Pi. Cartografierea este realizată de un senzor LIDAR de tip Hokuyo URG, iar măsurarea accelerației și a rotației este realizată de o unitate de măsurare inerțială (IMU). Zgomotul de măsurare a senzorului IMU sunt atenuate cu ajutorul unui filtru digital IIR (Infinite Impulse Response). Valoarea de orientare a robotului rezultată și datele extrase din norul de puncte LIDAR au fost fuzionate cu ajutorul algoritmului SLAM. În lucrarea de diplomă sunt prezentate datele măsurate și rezultatele obținute cu ajutorul algoritmilor. Am implementat și o interfață grafică responsabilă pentru configurarea și monitorizarea robotului, scrisă în Python și găzduită pe Raspberry.

Experimentele efectuate arată că metoda de măsurare dezvoltată se poate aplica bine la navigarea roboților mobili în prezența obstacolelor.

Cuvinte cheie: SLAM, măsurarea LIDAR, robotul mobil

**SAPIENTIA ERDÉLYI MAGYAR
TUDOMÁNYEGYETEM
MAROSVÁSÁRHELYI KAR
AUTOMATIKA ÉS ALKALMAZOTT INFORMATIKA SZAK**

**SLAM módszerek vizsgálata és
megvalósítása
DIPLOMADOLGOZAT**

Témavezetők:

Dr. Márton Lőrinc, egyetemi tanár

drd. Fehér Áron, tanársegéd

Végzős hallgató:

Krizbai Csaba

2021

Kivonat

Napjaink egyik népszerű kutatási területe az autonóm robotok és járművek környezetérzékelése. Környezetelemzéshez lézeres távolságmérőket (LIDAR) használhatunk, amelyek pontfelhők formájában rögzítik a környezetükben található akadályok térbeli helyzetét. Ezekkel a mérésekkel viszonylag pontos geometriai feltérképezést készíthetünk. Népszerűségét annak is köszönheti a LIDAR érzékelő, hogy a méréseit nem befolyásolja a külső megvilágítás.

A dolgozatom célja egy olyan mobilis robot megvalósítása, amely képes számára ismeretlen környezetet feltérképezni LIDAR érzékelő segítségével, majd az általa felmért helyszínen önállóan navigálni. Ezt a feladatot a SLAM (Simultaneous Localization And Mapping – Szimultán lokalizáció és feltérképezés) algoritmus segítségével oldottam meg. A robot adatfeldolgozó egységét egy Raspberry Pi egylemezes számítógép képezi. A feltérképezés egy Hokuyo URG típusú LIDAR szenzor, a gyorsulás és elfordulás mérés inerciális mérő (IMU) segítségével történik. Az IMU által szolgáltatott zajos értékeket végtelen impulzus válasz szűrő (Infinite Impulse Response - IIR) segítségével dolgoztam fel. Az ebből kapott robot orientációértéket és a LIDAR pontfelhőjéből kinyert adatokat fuzionáltam a SLAM algoritmus segítségével. A mért adatok és az algoritmusokból kapott eredmények megjelenítésre kerülnek. Megvalósítottam egy interfészt, amely robot konfigurálásáért, felügyeletéért felel, ami Python nyelven íródott és a Raspberry-n található.

Az elvégzett kísérletek azt mutatják, hogy a kidolgozott mérési módszer jól alkalmazható mobilis robotok navigációjához akadályok jelenlétében.

Kulcsszavak: lokalizáció, térképkészítés, lézeres távolságmérés, mobilis robot

Tartalomjegyzék

Ábrajegyzék.....	9
Egyenletek.....	10
Kódrészletek.....	10
Táblázatjegyzék.....	10
1. Bevezető.....	11
2. Elméleti megalapozás és felhasznált technológiák.....	12
2.1. Szakirodalmi tanulmány.....	12
2.2. Mobilis Robot.....	12
2.3. Beágyazott vezérlő számítógép.....	13
2.3.1. Raspberry Pi 4.....	13
2.4. Egyenáramú motor vezérlése.....	14
2.4.1. L292N motorvezérlő.....	14
2.5. Érzékelők.....	15
2.5.1. LIDAR érzékelő.....	15
2.5.2. Inerciális mérőegység.....	16
2.6. Kommunikációs protokollok.....	17
2.6.1. SCIP2.0.....	18
2.6.2. I ² C kommunikáció.....	20
2.7. Digitális adatfeldolgozási módszerek.....	21
2.7.1. SLAM.....	21
2.7.2. Párhuzamos adatfeldolgozás.....	22
2.7.3. Végtelen impulzusválasz szűrő.....	22
2.8. Ismert hasonló alkalmazások.....	23
3. A rendszer specifikációi és architektúrája.....	26
3.1. Hardver felépítése.....	26
3.1.1. Raspberry Pi 4.....	27
3.2. Szoftver tervezése és kivitelezése.....	28
3.2.1. LIDAR érzékelővel való kommunikáció.....	28
3.2.2. IMU mérés és kalibrálás.....	31
3.2.3. Motor szabályozása.....	37
3.2.4. SLAM algoritmus.....	38
3.2.5. Mobilis robot szoftvere.....	42
4. Üzembe helyezés és kísérleti eredmények.....	47
4.1. IMU mérések.....	47
4.2. Motor szabályozás mérése.....	47
4.3. SLAM mérések.....	49
4.4. Felmerült problémák és megoldásaik.....	53
4.4.1. IMU problémák.....	53
4.4.2. Számításkapacitás.....	55
4.4.3. LIDAR érzékelő hibái.....	55
5. Következtetések.....	57
5.1. Megvalósítások.....	57
5.2. Hasonló rendszerekkel való összehasonlítás.....	57
5.2.1. Xiaomi Mi Robot okosporszívóval való összehasonlítás.....	57
5.3. Továbbfejlesztési lehetőségek.....	57
6. Köszönetnyilvánítás.....	58
7. Irodalomjegyzék.....	59

Ábrajegyzék

1. ábra: H - híd.....	14
2. ábra: Hokuyo LIDAR érzékelő látószöge	16
3. ábra: GY - 91 IMU érzékelő	17
4. ábra: SPIC2.0 kommunikációs protokoll	19
5. ábra: MD parancs protokollja (Gazda - Érzékelő)	20
6. ábra: MD parancs protokollja (Érzékelő - Gazda)	20
7. ábra: I2C kommunikációs protokoll.....	21
8. ábra: MPU9250 kommunikációs protokollja	21
9. ábra: Szűrők karakterisztikái	23
10. ábra: Okosporszívó telefonos applikációja	24
11. ábra: Megvalósított mobilis robot	26
12. ábra: Mobilis robot tömbvázlata	27
13. ábra: Raspberry Pi 4 lábkiosztása.....	28
14. ábra: UrgBenri szoftver.....	29
15. ábra: LIDAR mérések kirajzolása	31
16. ábra: IMU mérések.....	34
17. ábra: IMU kalibráció	35
18. ábra: IMU szoftver aktivitás diagramja.....	36
19. ábra: Motor vezérlő szoftver kezelőfelülete.....	37
20. ábra: Mobilis robot szabályozási diagramja.....	38
21. ábra: Térkép szerkezete	39
22. ábra: Bresenham algoritmus példa	40
23. ábra: Matlab példa részecskeszűrőre	41
24. ábra: Monte Carlo lokalizáció	42
25. ábra: Mobilis robot kezelőfelülete.....	43
26. ábra: Folyamatábra	44
27. ábra: Mobilis robot aktivitás diagramja	45
28. ábra: Mobilis robot osztálydiagramja.....	46
29. ábra: Szűrők összehasonlítása	47
30. ábra: Referencia és mért szög.....	48
31. ábra: Beavatkozó jel	48

32. ábra: Referencia és mért szög.....	49
33. ábra: Beavatkozó jel	49
34. ábra: Térkép készítés, mérések 1.....	50
35. ábra: Térkép készítés, mérések 2.....	51
36. ábra: Térkép készítés, mérések 3.....	52
37. ábra: Térkép készítés, mérések 4.....	53
38. ábra: Zajos IMU mérések1	54
39. ábra: Zajos IMU mérések2	54
40. ábra: Folyamatfigyelő alkalmazás.....	55
41. ábra: Zajos SLAM mérések.....	56

Egyenletek

1. egyenlet: X és Y pont meghatározása	30
2. egyenlet: Bemenetek skálázása	32
3. egyenlet: Tengely kompenzálás	32
4. egyenlet: Torzítási hiba	32
5. egyenlet: Lágyvas torzítás	33
6. egyenlet: Magnetométer modellje	33
7. egyenlet: magnetométer modellje kifejezve.....	33
8. egyenlet: Mágneses mező.....	33
9. egyenlet: Behelyettesítve a magnetométer modelljébe	33
10. egyenlet: Kvadratikus formában	34
11. egyenlet: távolság számolás	40
12. egyenlet: közelebbi pont kiszámolása1	40
13. egyenlet: közelebbi pont kiszámolása2	41

Kódrészletek

1. kódrészlet: LIDAR mérések dekódolása.....	30
2. kódrészlet: A és b mátrix számolása	34
3. kódrészlet: IIR szűrő megvalósítása Python nyelven.....	37
4. kódrészlet: Motor szabályzó Python nyelven.....	38

Táblázatjegyzék

1. táblázat: Többprocesszoros és többszálás adatfeldolgozás	44
---	----

1. Bevezető

Az ember érzékei, látás, tapintás, szaglás szoros kapcsolatban állnak a tudásával. Ezért számára egy idegen hely feltérképezése és egy mentális térkép kialakítása igencsak triviális feladat. Alapvetően egy robot, valamint egy ember fő navigációs feladatai hasonlóak lehetnek. Az ember érzékszerveit különböző szenzorok, valamint a gondolkodását egy vagy több mikrovezérlő jelképezheti. Ezekből az összefüggésekből következtethetünk arra, hogy egy robot számára sem jelenhet problémát egy idegen hely feltérképezése. A feladat megoldására, már az 1986-ban megszervezett IEEE Robotika és Automatizálási Konferencián mutattak be megoldásokat. Viszont a XXI. században az autonóm, önvezető rendszerek jelentős változáson, fejlődésen mentek keresztül és egyre nagyobb népszerűségnek örvendhetnek.

A célom egy olyan autonóm robot megvalósítása, amely képes saját környezetét feltérképezni, abban a saját helyzetét meghatározni és önállóan, ütközésmentesen navigálni. A robot helyzetének meghatározásához és könnyed navigáció érdekében zárt térben lesz alkalmazható. Ez a három probléma: térképkészítés, helymeghatározás és úttervezés átfedésben vannak egymással. Ezekre a problémákra megoldást a SLAM (Simultaneous localization and mapping - Szimultán lokalizáció és feltérképezés) algoritmus nyújt. Ennek az algoritmusnak az alapjait szeretném jobban megismerni és alkalmazni. Dolgozatomban az alapfogalmak letisztázása után, szeretném bemutatni az általam elkészített mobilis robotot, majd az alkalmazott SLAM algoritmust.

Választásom azért erre a területre esett, mert mindig kíváncsivá tett az autonóm járművek témaköre. Betekintést szerettem volna nyerni, hogy milyen algoritmusok és érzékelő által határozzák meg a pozíciójukat egy ismeretlen környezetben. A témaválasztásomat az is indokolja, hogy a mobilis robotok iparban való alkalmazása egyre népszerűbb, de akár a hétköznapiakban is egyre több robottal találkozhatunk. Szerettem volna jobban belelátni a robotok szerkezeti felépítésében, valamint az irányítási rendszerükbe.

2. Elméleti megalapozás és felhasznált technológiák

2.1. Szakirodalmi tanulmány

Annak érdekében, hogy egy pontos tervet tudjak kialakítani a SLAM algoritmusról, valamint a mobilis robotok navigációról, első sorban hasonló megvalósítások után kutattam. A hasonló témájú publikációkból következtetéseket próbáltam levonni és aszerint megvalósítani a saját projektem. A következőkben három olyan publikációt szeretnék bemutatni, amely részben kötődik az én témámhoz is.

- **Lidar alapú 2D térképek összehasonlítása:**

Ebben a publikációban [1] a vizuális SLAM és lézeres SLAM algoritmus kerülnek összehasonlításra. A lézeralapúérzékelő használatakor fentáll a robot „elvesztési” esélye túlságosan nyílt terepen, például szabadtéren, mivel a robot nincs mihez viszonyítsa a helyzetét. A szerzők a térképkészítésnél görbületi és piramis szűrőket alkalmaztak, valamint implemtálták a Gmapping és ORBSLAM2 algoritmust a lézer és vizuális SLAM mellé.

Kísérleti eredmények azt mutatják, hogy a lokalizációs hiba a valós távolsághoz viszonyítva kevesebb lehet mint 5%.

- **Sztereo látás és térképezés szinkronizálatlan kamerákkal:**

A projekt [2] célkitűzése hasonló az én témámhoz, egy olyan mobilisrobotot szeretne megvalósítani, amely képes környezetét feltérképezni és így navigálni abban. Ebben a dolgozatban IMU érzékelőt, mono és sztereó kamerarendszerekkel alkalmaztak a térkép elkészítésénél. [3]

- **Betekintés a mobilis robot navigációba:**

Ebben a tanulmányban [3] az elmúlt 20 év mobilis robot navigációjához szükséges fejlesztések vannak bemutatva. Mind a beltéri, mind a kültéri navigáció bemutatásra kerül a cikkben. A beltéri navigáció fejezetben háromfajta navigációt tárgyal: Térképalapú, Térképkészítésen alapuló és térképnélküli navigáció.

2.2. Mobilis Robot

Robot szó régi szláv nyelvből ered, jelentése szolgaság (rabota). Alapvetően a robotokat olyan feladatok elvégzésére is használjuk, amelyeket kellő pontossággal, gyorsasággal kell elvégezni.[4] Mobilis robotnak nevezzük, azokat a robotokat, amelyek képesek a környezetükben való helyváltoztatásra. A robotokat általában kerekekkel szerelik fel és különböző villamosgépek segítségével hajtják meg. Az energiaellátásért általában akkumulátorok felelenek. Feladatuk része,

hogy eljussanak a kívánt célpontba anélkül, hogy bármiféle károsodást okozzanak a környezetükben, valamint magukban. Erre a feladatra három különböző megoldást alkalmazhatunk: robot távezérlése, egy előre létrehozott pálya követése és robot autonóm mozgása. Az autonóm mozgáshoz teljesítéshez a robotnak rendelkeznie kell érzékelőkkel, melyek segítségével megtudják határozni a lokalizációjukat egy referenciakeretben. A robot korodinált mozgásáért, érzékelőktől való adatgyűjtésért egy központi számítógép fele. Az útvonaltervezés csak akkor jöhet létre, ha a robot tudja a saját helyzetét és a célpontot koordinátáit egy adatott referenciakeretben. Számos navigáció típus létezik: látásalapú, inerciális, akusztikus és rádió navigációs. Ebben a dolgozatban egy látásalapú navigáció megvalósítása a cél, ezért ennek a bemutatására térek ki. Látásalapú navigációban kamerákat és lézeres szenzorokat használnak a lokalizáció meghatározásához. A legegyszerűbb, ha zárt térben végezzük a navigációt, mivel így a robotunk könnyen tudja a pozícióját viszonyítani az akadályokhoz, falakhoz. [5]

2.3. Beágyazott vezérlő számítógép

A mobilis robot feldolgozóegységét általában mikroprocesszor, beágyazott mikrovezérlő vagy személyi számítógép képezi. Feldolgozóegységet az elvégzendő feladat komplexitása és számításigényesége választja. Jelen esetben egy mikroszámítógép képezi a mobilis robot feldolgozóegységét. Egyetlen áramkörbe integrált mikroprocesszor, memória és ki/bemeneti portok alkotják a mikroszámítógépet. Gyakran statikus memóriákkal és olcsó 8 vagy 16 bites processzorokkal szerelik őket. Általában egy MicroSD vagy eMMC kártyafoglalattal látják el, mivel az operációsrendszer külső memóriakártyákon található. Legtöbb modell rendelkezik USB, különböző audio és videó csatlakozóval. Ellentétben az asztali számítógépekkel, ezek az eszközök nem rendelkeznek bővítőkártyák foglalattal. [6]

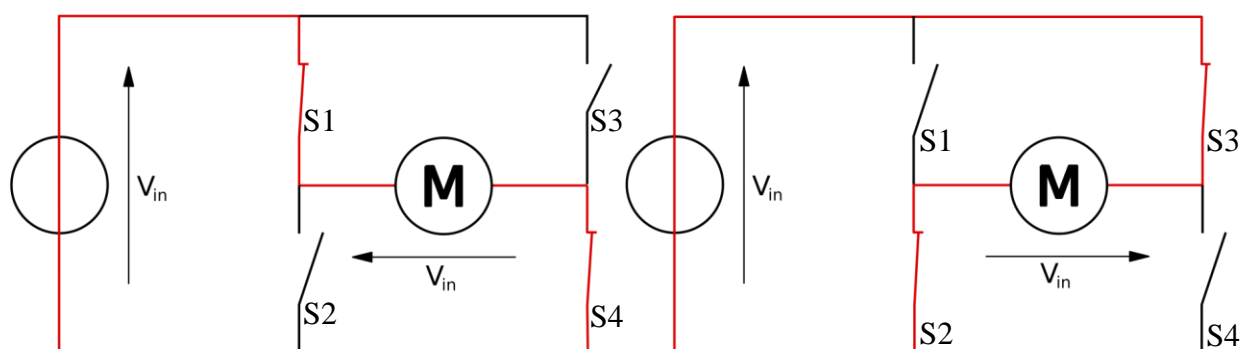
2.3.1. Raspberry Pi 4

A robot feldolgozóegységét egy Raspberry Pi 4 egylemezes számítógép képezi. Ezt a kis mikroszámítógépet az Egyesült Királyságban fejlesztettek oktatási célokra, de most már világszerte népszerűvé vált és nem csak oktatási célokra használják. A számítógépre többféle, többnyire Linux alapú operációs rendszert választhatunk, például: Raspbian, Pidora, Minibian, ROS, stb... Ez a termék a Broadcom cég BCM2837B0 processzora köré épül, ami egy 64 bites ARM Cortex-A72 1,5GHz-es processzor. Alapvetően 1,5GHz működik a CPU, viszont hogyha a processzor eléri a 70 °C, akkor teljesítmény csökkenés történik, ugyanis az órajel automatikusan 1,2GHz alá esik. A Raspberry Pi 4 1GB, 4GB vagy 8GB LPDDR4-es memóriával is kapható. Ez a kis számítógép vezeték nélküli LAN-t és 5.0 Bluetooth kommunikációra is képes, valamint

rendelkezik két darab USB 2.0 és két darab 3.0 csatlakozóval, két darab micro - HDMI videókimenettel, 3.5 milliméteres sztereó jack csatlakozóval és egy Gigabit Ethernet csatlakozóval. A Pi 4-as nagyobb teljesítményigénnyel működik, mint a korábbi társai, szóval egy 2,5 Amperes, USB Type-C csatlakozó tápegység szükséges a működtetéséhez. Tápellátást a AC DC adapter látja el, ami maximum 15.3W teljesítményt tud leadni 5.1V-on. A Raspberry Pi 4 jelenlegi piaci ára körülbelül 30€. [7]

2.4. Egyenáramú motor vezérlése

Ha a villanymotor szénkefejére villamos energiát kötünk, akkor a motor tengelye adott sebességgel forogni kezd egy irányba. A forgásirányát és sebességet egy H hídnak nevezett motorvezérlőáramkör segítségével tudjuk vezélni. A H-híd elvi rajza látható a következő 1. ábrán. Az ábrán látható, hogyha a S1 és S4 kapcsoló zárva van, akkor a motor jobbra forog, ellenkező esetben, ha az S2 és S3 van zárva, balra forog. Ezeket a kapcsolókat általában félvezetők alkotják, amelyeket impulzusszélesség-moduláció (Pulse width modulation – PWM) segítségével vezérelnek meg. Így nemcsak a forgási irány, hanem a forgási sebesség is állítható. [8]



1. ábra: H - híd

2.4.1. L292N motorvezérlő

Az egyenáramúmotor vezérlését egy L292N H – híd típusú, Multiwatt15 tokozású motorvezérlő végzi. Ezt az áramkör kettő teljes H híddal rendelkezik, amelyet úgy terveztek, hogy szabványos TTL (tranzisztor – tranzisztor logika) szinteket fogadjon. Így akár két egyenáramú motort is megvezérelhetünk különböző sebességgel és forgásiránnyal. A vezérlő képes a DC motorokat 5V-tól egészen 35V-ig kezelni 2A-es csúcsáram mellett. Ez a modul három csavaros sorkapoccsal rendelkezik, amelyből kettő a motor betáplálására szolgál és a harmadik pedig a motorok vezérlésére. Az L29N motorvezérlő jelenlegi piaci ára körülbelül 5€. [9]

2.5. Érzékelők

Az érzékelők legáltalánosabb meghatározása szerint olyan eszköz, modul, gép vagy alrendszer, amely a bekövetkező eseményeket észleli és elküldi egy más elektronikai eszköznek, gyakran számítógépes processzornak. [10]

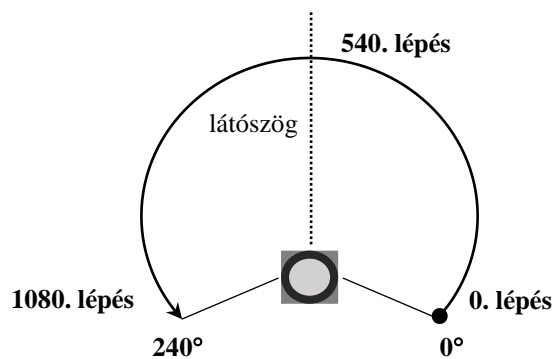
2.5.1. LIDAR érzékelő

A lézert egy koncentrált fénysugárként lehet jellemezni. A lézerből távozó fény időben és térben koherens, valamint bizonyos hullámhosszall rendelkezik. Az emberi szem a 380 és 750 nm (nanométer) hullámhosszú elektromágneses sugárzást láthat, ami frekvenciatartományban 400-749 THz (terahertz). Ez az érték felett lévő frekvenciatartomány az ember számára már nem látható. Például az ultraibolya hullámhossz tartománya kisebb, ezért nem látható szabad szemmel. A lézert számos helyen alkalmazzák. Komplexebb műtéteket lézer segítségével hajtják végre, valamint lézernyalábbal mérik fel a terepet az autonóm járművek számára.[11]

A lézerradar vagyis a LIDAR (Light Detection and Ranging) a távolságok meghatározására szolgáló módszer. A lézerradar három fő egységből tevődik össze: egy lézeres adóból egy optikai érzékelőből és egy forgó tükörből. A tükör szerepe, hogy a lézer egy vagy kétdimenzióban is képes legyen a környezetet letapogatni. Az adó ultraibolyás fényt bocsájt ki, amelyet, az optikai érzékelő fogad. Általában 600 – 1000 nm hullámhosszú lézereket alkalmaznak. Katonai alkalmazásban akár 1550 nm lézerrel is találkozhatunk, az ilyen hullámhosszú sugarak még éjjellátó szemüveg segítségével sem látható. A kiküldött impulzusok késve érkeznek vissza, így a köztes időből megállapítható a tárgy távolsága. Távolság számoláskor figyelembe kell venni a közeget is, mivel a fény más sebességgel terjed a vákuumban, mint például a vízben. A lézer fizikai tulajdonsága miatt korlátolt a mérőtávolsága, általában minden gyártó megadja, hogy milyen távolságot képes bemérni az érzékelő.[12]

2.5.1.1 Hokuyo URG

Térképezéshez szükség volt egy LIDAR érzékelőre, amely kellő gyorsasággal és pontossággal feltudja mérni a robot környezetét. Egy Hokuyo Urg 04LX-UG01 LIDAR szenzort használtam a robot környezetének feltérképezéséhez. A LIDAR az ultraibolya lézerszkennelés segítségével határozza meg a környezetében lévő tárgyak közelségét. A szenzor mindössze 160g és viszonylag kis energiafelvétellel rendelkezik, a gyártó szerint 2.5W, ezért is tökéletes választás robotokra. A lézerradar házának mérete 50x50x70 mm és IP40 védettséggel rendelkezik. Ahogyan a 2. ábra is illusztrálja, a látószöge korlátolt, összesen 240° tud feltérképezni, valamint 56000



2. ábra: Hokuyo LIDAR érzékelő látószöge

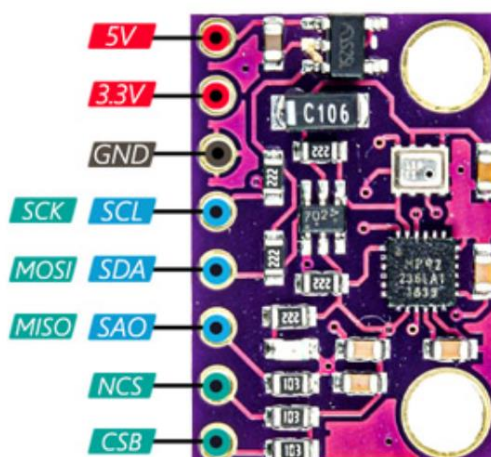
milliméter a maximális látótávolsága. A 240° fokot mindössze 1080 lépésben olvassa le, ebből egy egyszerű osztással kideríthető, hogy 0.2° a kihagyása. Ezek az értékek mellett kellő pontosan tud mérni, mindössze ± 30 milliméter a tévedhet. A betáplálási feszültsége 5VDC és az áramfelvétele 500mA. Egy teljes feltérképezés 100 ms alatt történik meg. A Hokuyo URG RS232 és USB interfésszel van ellátva. Az USB interfész CDC szabványt használ, ezért a kommunikációs protokoll hasonló az RS232C-hez. A szenzor és a gazda(host) között SCIP kommunikációs protokollon keresztül történik a kommunikáció. Ezt a protokollt Japán kutatócsoport fejlesztette ki az URG típusú érzékelők számára. Ennek a segítségével a kommunikáció nagyon rugalmas és könnyen implementálható. Az URG 04-LX LIDAR már támogatja a SCIP 2.0 kommunikációs protokollt, de indításkor alapértelmezetten az SCIP1.1 – ből indul. Ennek az érzékelőnek a jelenlegi piaca ára körülbelül 800€ [13].

2.5.2. Inerciális mérőegység

Az inerciális mérőegység (IMU - internal measurement unit) olyan elektronikus eszköz, amely segítségével megmérhetjük egy test szögsebességét, gyorsulását és orientációját egy inerciarendszerhez képest. A szögsebességet a giroszkóppal, a gyorsulást gyorsulásmérővel és az orientációt magnetométer segítségével tudjuk mérni. Az IMU-k navigációhoz történő használatának fő hátránya, hogy az érzékelők nagyon érzékenyek a zajokra, így alkalmazás során kompenzálni kell, hogy valós mérési eredményeket kaphassunk. [14]

2.5.2.1 GY - 91

A mobilis robot orientációjának meghatározáshoz kilenc szabadságfokú (DOF) IMU használtam, amely GY - 91 névre hallgat. A 3. ábrán érzékelő látható, mindössze 20 mm hosszú és 15 mm széles, a kompakt mérete miatt is kiváló választás robotokra.



3. ábra: GY - 91 IMU érzékelő

Ezen az érzékelőcsoporton két darab több áramkörből álló modul (MCM – multichip modul) található meg, egy MPU9250 és egy BMP280 -as. Az MPU9250 -ben, a giroszkóp és gyorsulásmérő mellett még megtalálható magnetométer is, AK8963 tokozásban. Ezek az érzékelők mindössze kilenc szabadságfokot biztosítanak. Ebben a modulban a giroszkóp, gyorsulás, valamint a magnetométernek három darab 16 bites analóg digitális konverter (ADC) biztosítja a kimeneteik mintavételezését. Különböző mozgás lekövetéséhez a felhasználó által programozható áramköri elemeket is tartalmaz, így a giroszkóp, a gyorsulásmérő és a magnetométer mérési tartománya bővíthetőek. A 16 bites giroszkóp skálázási tartománya ± 250 , ± 500 , ± 1000 vagy ± 2000 (dps) fok/másodperc, a gyorsulásmérő tartománya $\pm 2g$, $\pm 4g$, $\pm 8g$ vagy $\pm 16g$ és a magnetométer tartománya egészen $\pm 4800\mu T$ állítható. A BMP280 -as modulban egy barométer található, ami nyomásmagasság mérésére alkalmas érzékelő. Az érzékelőt összesen nyolc kimentő lábbal látták el, amiből három az energiaellátásért felel, kettő a 400kHz – es I²C kommunikációért, a maradék pedig az 1Mhz -es SPI kommunikációért. Az előző ábrán kék színnel jelölt lábak az I²C kommunikációra és a türkíz színnel jelölt lábak pedig az SPI kommunikációra használhatóak. [15]

2.6. Kommunikációs protokollok

A kommunikációs protokoll olyan megalapozott szabályok, amelyek szerint megvalósulhat az adatcsere egy mester és egy szolga eszköz között. Egyszerűbb feladatokhoz egyszerűbb és nehezebb feladathoz bonyolultabb protokollt használunk. A protokoll feladata, hogy adatvesztés nélkül küldjön vagy fogadjon adatokat, valamint, hogy az adatokat helyes sorrendbe küldje el. Két eszköz közti kommunikáció két nagy kategóriába tagolható: szinkron és aszinkron

átvitel. Az egyik fő különbség a, hogy a szinkron átvitelhez szinkronizált órajel szükséges. Az aszinkron átvitelnél pedig stop és start biteket használunk.

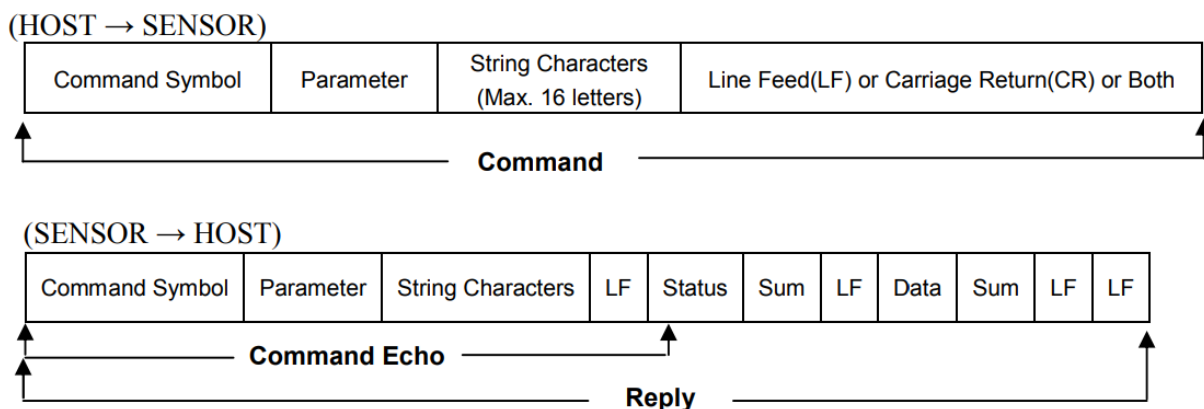
A szinkron kommunikáció lehetővé teszi a párhuzamos, mindkét irányába való kommunikációt, vagyis duplex módban történik az adatcsere. Szinkronizáló biteket használunk, így a vevő tudja, hogy hol kezdődik az új byte. A szinkron átvitel nagy mennyiségű adat átvitelére alkalmas. Valós idejű kommunikációt tesz lehetővé. Csevegőszobák, videokonferenciák és telefonos beszélgetések kommunikációjánál gyakran alkalmazzák, mivel nagyon hatékonyan a nagy adatok átvitelénél.

Az aszinkron kommunikáció fél-duplex üzemmódban működik, vagyis mindkét fél képes kommunikálni egymással, csak nem egyidejűleg. Általában a 8 bites adat mellé paritásbit adódik. A szinkronizációt ezek segítségével oldódik meg. Ez a kommunikáció gazdaságos, mivel nem igényel kétutas kommunikációt. [16]

2.6.1. SCIP2.0

A 1995-ben a Compaq, DEC, IBM, Intel, Microsoft, NEC, Norther Telecom vállalatok összefogásával alakult meg az USB (Universal Serial Bus) első verziója. Azóta számos verziót fejlesztettek ki, legújabb változata az USB 3.1. Az USB 2.0 kommunikációs osztályba sorolja az USB eszközöket. Virtuális soros porton megvalósító USB kommunikáció a CDC (Character Device Class) eszközosztályba tartozik. Ez a kapcsolat a PC szemszögéből virtuális soros portnak látszik. Az SCIP2.0 USB CDC eszközosztályt alkalmaz a kommunikáció megvalósításáért. A SCIP2.0 szabványt egy Japán kutatócsoport fejlesztette ki (Intelligent Robot Laboratory, University of Tsukuba), azzal a céllal, rugalmas és hatékony interfészt biztosítson robotikai alkalmazásokhoz. Ez a protokoll egy előre megszabott karakterláncot vár a gazdától. Mivel a Hokuyo URG támogatja az SCIP2.0, ezért a következőben az érzékelő és a gazda közötti kommunikációt szeretném bemutatni.

Hogyha az érzékelőhez megfelelő parancs érkezik, akkor válaszként visszaküldi a küldött parancs visszhangját, egy állapotot jelző karaktert, egy összeg, a parancshoz tartozó adatot és még egy összeget. A kommunikációs protokoll a 4. ábrán látható.



4. ábra: SPIC2.0 kommunikációs protokoll

Az érzékelő adatai kódolva vannak, hogy csökkentsék a fogadó és érzékelő közötti átviteli időt. Ezért ezeket az adatokat feldolgozás előtt dekódolni kell. Erre három technikát biztosít számunkra a gyártó: két karakteres kódolás, három karakteres kódolás és négy karakteres kódolás. Én a két karakteres kódolást alkalmaztam, ezért erre szeretnék kitérni a következőkben. Ez a kódolási technika legfeljebb 12bit hosszúságú adatok kifejezésére alkalmazták. A kódolás úgy történik, hogy a beérkező adatokat hat felső és hat alsó bitre bontják, majd 30 Hexát adnak hozzá, hogy ASCII karakterré alakítsák őket. A többi kódolási technika is hasonló képen működik. Összesen hét fajta, előre definiált parancs (Command Symbol) van implementálva a SCIP 2.0 verzióban:

- Három fajta érzékelő paraméter lekérdező parancs (VV, PP, II).
- Mérési ki/be kapcsoló parancs (BM, QT).
- RS232C bitsebessége beállítása (SS).
- Két fajta mérési érték beolvasása (MDMS, GDGS).
- Motor sebesség beállítása (CR).
- Időbélyeg beállítása (TM).
- Újraindító parancs (RS).

Ezek a parancsok közül a legfontosabbat szeretném kiemelni. Az *MDMS* parancs segítségével tudjuk az érzékelőtől begyűjteni a mért távolságokat. Két típus közül tudunk választani: *MD* vagy *MS*. Az *MD* a három karakterrel kódolt adatokra, az *MS* pedig a két karakterrel kódolt adatokra vonatkozik. Szükséges még hat paraméter megadása is, amellyel a mérési paramétereket tudjuk szabályozni. A “Startin Step” és az “End Step” paraméterek segítségével tudjuk meghatározni, hogy milyen sugárba végezze a méréseket. A “Cluster Count” paraméterrel tudjuk egyetlen adattá egyesíteni a méréseket. Például hogyha 3-ra állítjuk, és a mért értékek: 3095, 3055 és 3062, akkor

az érzékelőtől 3055 értéket fogjuk megkapni. Az MDMS parancs protokollja az 5. és 6. ábrán látható.

M (4dH)	D (44H) or S (53H)	Starting Step (4bytes)	End Step (4 bytes)	Cluster Count (2bytes)	Scan Interval (1 byte)
Number of Scans (2 bytes)	String Characters	LF			

5. ábra: MD parancs protokollja (Gazda - Érzékelő)

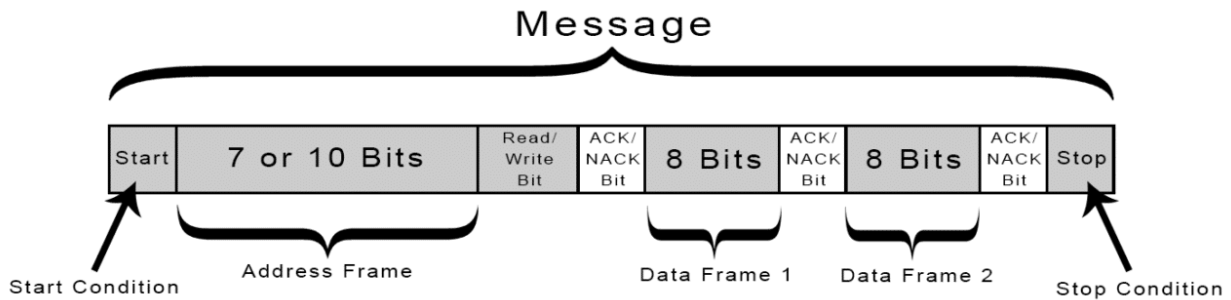
M	D or S		Starting Step		End Step		Cluster Count		Scan Interval	
Remaining Scans			LF							
9	9	b	LF		Time Stamp (4byte)			Sum	LF	
Data Block 1 (64 byte)				Sum		LF				
-----				Sum		LF				
Data Block N (64 byte)				Sum		LF		LF		

6. ábra: MD parancs protokollja (Érzékelő - Gazda)

A válaszként kapott adatcsomagban található STATUS mezővel tudjuk ellenőrizni, hogy a mérés sikeres volt, vagy sem. Ezek mellett az érzékelő egy 24 bites belső időzítővel rendelkezik, így minden mérés 0. lépésénél megkapjuk a 4 byte - os időbélyeget.[12]

2.6.2. I²C kommunikáció

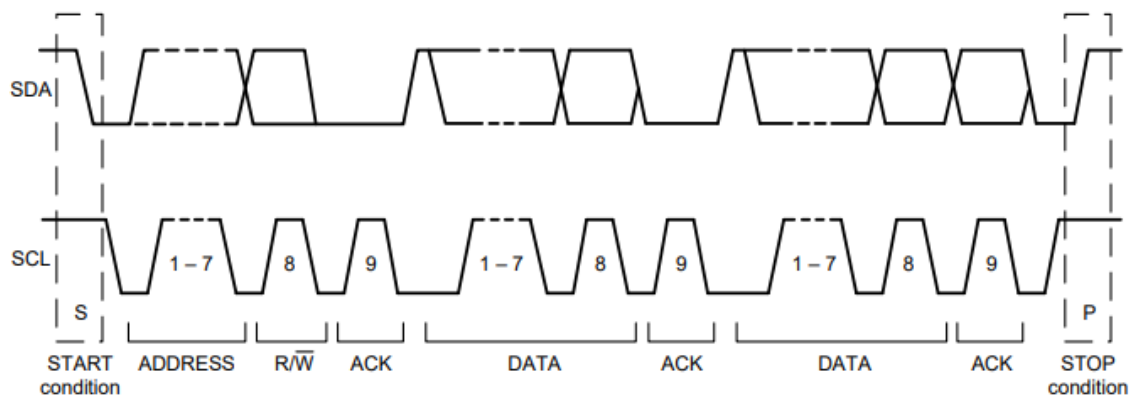
Az inter-IC (I²C) -t eredetileg a Philips fejlesztette ki a saját fejlesztésű integrált áramköreihez. A kommunikációs protokoll előnye, hogy megengedi több szolga (slave) kommunikáljon egy vagy több mesterrel (master). Egy szolga és a mester közti kommunikáció kialakításához összesen két darab vezeték szükséges. Az egyik vezetéken a mester és szolga közti adatcsere valósul meg, ezt SDA (Serial Data) – val jelöljük. A másik vezetéken az órajel található, vagyis az SCL (Serial Clock). Az SPI – hez hasonlóan az I²C is szinkron kommunikációs protokoll, vagyis a mester és szolga közti adatcserét az órajel szinkronizálja. Az üzenetek keretbe vannak zárva, start és stop mezők által. Minden üzenetnek van egy címmezője, amely a szolga bináris címét tartalmazza és egy vagy több mező, amely megadja az adatot tartalmazza. Általában az adatmezőket nyugtázó bit követi, hogy a fogadó fél meggyőződhessen az üzenet helyességéről [16]. A 7. ábrán az üzenet protokollja látható:



7. ábra: I2C kommunikációs protokoll

2.6.2.1 GY – 91érzékelővel való kommunikáció

A kommunikáció a buszon akkor kezdődik, amikor a mester a start állapot a buszra helyezi. Ez azt jelenti, hogy adatvonalat (SDA) magasról, alacsony szintre állítja. Ameddig a SCL vonal magas állapotban van, addig a vonal foglalt. Miután a kommunikáció elkezdődött, a mester kiküld egy 7 bites címet (Slave Address = 0x68), amit követ egy 8 bites írás vagy olvasás jelző cím. Ezután a mester felszabadítja az adatvonalat, nyugtázó jelet (ACK) várva az érzékelőtől. Minden adatot nyugtázó bitnek kell követnie. A nyugtázáskor a szolga eszköz az adatvonalat alacsony szintre húzza, mindaddig ameddig SCL vonal magas állapotban van. Az adatátvitelt a mester STOP – állapottal zárhatja be, ami azt jelenti, hogy az adatvonalat magasszintre állítja [14]. A 8. ábrán a mester és szolga közti kommunikáció látható:



8. ábra: MPU9250 kommunikációs protokollja

2.7. Digitális adatfeldolgozási módszerek

2.7.1. SLAM

A közzétett SLAM megközelítéseket önjáró autókban, pilóta nélküli légi járművekben, autonóm víz alatti járművekben és még az emberi testben is alkalmazzák [18]. Alapvetően a SLAM problémára két különböző megoldást alkalmaznak. Az egyik gyakori megoldás a vizuális

SLAM, ahol különböző kamerák segítségével próbálják feltérképezni a környezetet. A vizuális SLAM olcsó kamerák alkalmazásával viszonylag olcsón megvalósítható. Valamint a kamerákból nyert információkat tárgyfelismerésre is felhasználható. A másik megoldás a lézer alapú távérzékelő alkalmazása, vagyis LIDAR SLAM. A lézeres érzékelők jóval pontosabb és gyorsabb mérésekre képesek, mint a kamerák. Ezért akár mozgó járműveken is alkalmazhatóak. A LIDAR kimeneti adatai 2D vagy 3D pontfelhő is lehet. Általában a pontfelhők összeillesztésével becsüljük meg a megtett távolságot és a megtett távolságból számoljuk ki a robot lokalizációját. Népszerű közelítő megoldások közé tartozik a részecskeszűrő (másnéven Monte Carlo módszerek) és kiterjesztett Kalman Filter algoritmus használata. A pontfelhő-egyeztetés általában nagy feldolgozási teljesítmény igényel, a feldolgozási sebesség javítása érdekében optimalizálni szokták a folyamatot más érzékelők bevezetésével. Például kerék fordulatszáma mérése, globális navigációs műholdas rendszer (GNSS), inerciális mérőegység (IMU). [19]

2.7.2. Párhuzamos adatfeldolgozás

Az adatok feldolgozásának sebessége növelése érdekében gyakran alkalmaznak párhuzamos adatfeldolgozást a számítástechnikai rendszerek világában. A párhuzamosításnak ötlete azon a tényen alapul, hogy számos olyan nagy feladatok léteznek, amit több kisebb alrészre lehet leosztani, amit majd a feldolgozó egység egyszerre elvégezhet. Ezeket az alrészeket szükséges összehangolni, szinkronizálni. A párhuzamos adatfeldolgozás megvalósítása jóval bonyolultabb, mint az egymás követő, mivel a párhuzamos rendszereknél az erőforrás menedzselés is szükséges. A folyamatok az erőforrásért versengenek, így számos hiba felmerülhet.

A párhuzamosan végződő folyamatok közti kommunikációra a legjobb megoldás az üzenetsorok alkalmazása. Az üzenetsor FIFO (First – In, First - Out) elven működő láncolt lista, ami azt jelenti, hogy azt első beérkező üzenetet tudjuk elsőnek kiolvasni. Mivel lista, ezért rendelkezik egy mérettel. [20]

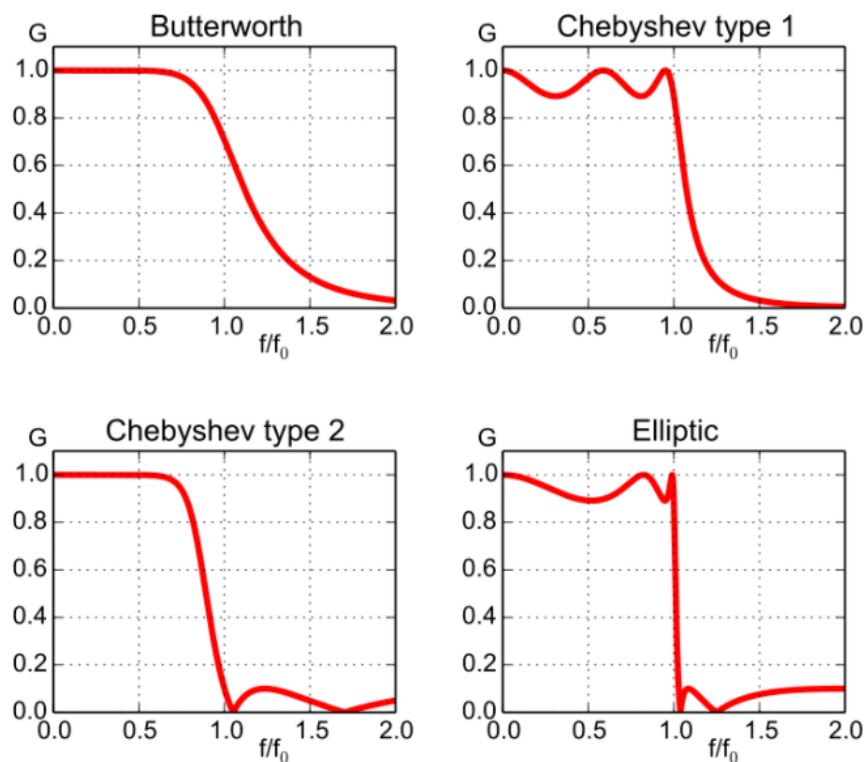
2.7.3. Végtelen impulzusválasz szűrő

A végtelen impulzusválasz szűrő (IIR) a digitális szűrők családjába tartozik [21]. A végtelen impulzusválasz egy olyan tulajdonság, amely számos lineáris időinvariáns rendszerre vonatkozik. Jelentése, hogy az impulzusválasza egy bizonyos pontot túl nem válik nullává, csak közelíti azt és a végtelenségig folytatódik. A véges impulzusválaszú szűrő (FIR) pontosan az ellentéte, mivel egy T idő után pontosan nullává válik. Folytonos (analóg) szűrők mintavételezett változata. A szűrő impulzus válaszfüggvénye véges, időben invariáns és pólusokat is tartalmaz,

ezért stabilitásvizsgálat szükséges. A klasszikus IIR szűrő különböző módszerekkel közelítik meg az ideális aluláteresztő szűrőt. Ezek a módszerek a következők:

- **Butterworth** szűrő rendelkezik a legsimább, hullámzásmentes görbével mind az áteresztési és vágási tartományban, azonban lassúak az átmenet tartományban.
- **Chebyshev** szűrők karakterisztikája gyorsabb, mint a Butterworthé, viszont a hullámmal rendelkeznek az áteresztő (Chebyshev I) vagy záró tartományban (Chebyshev II).
- **Bessel** szűrő hasonlít a Butterworth típushoz, azonban átmeneti sebessége nagyon lassú, mivel egy egyszerű RC szűrőhöz hasonlít.
- **Elliptikus** szűrő rendelkezik a legmeredekebb átmenettel, viszont kiegyenlített hullámmal rendelkezik mind az áteresztő, mind a záró tartományban. Ez a típus a Chebyshev keveréke.

A 9. ábrán szűrők tulajdonságai láthatóak:



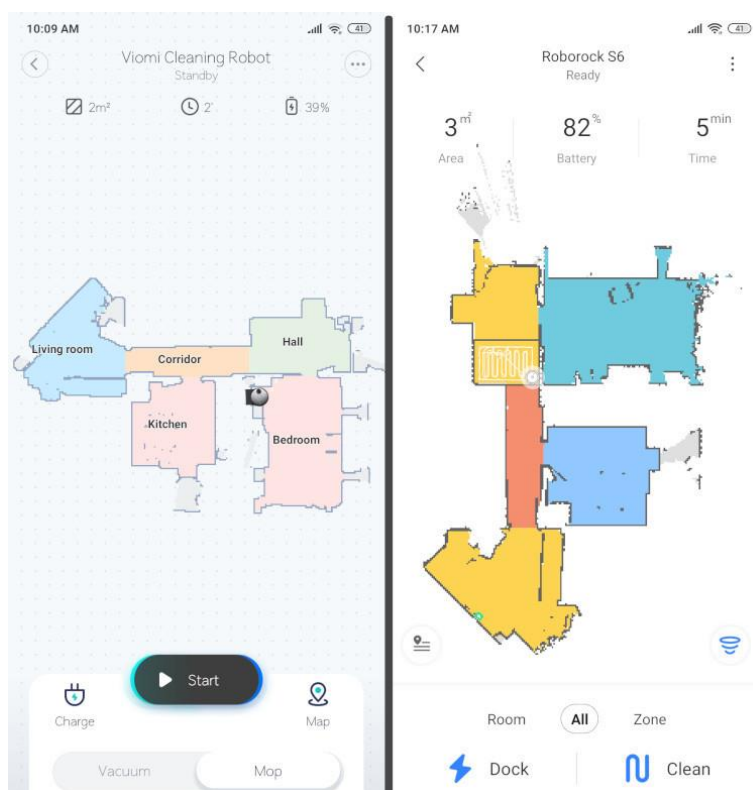
9. ábra: Szűrők karakterisztikái

2.8. Ismert hasonló alkalmazások

Környezetünkben egyre több készülék, jármű használ már SLAM algoritmusokat. Felhasználási területe nagyon szerteágazó, számos helyen alkalmazzák ezt az algoritmust.

- **Xiaomi Mi Robot okosporszívó**

A Xiaomi kínai multinacionális elektronikai cég, amely 2010-ben alakult. Elsősorban mobiltelefonokat és mobilalkalmazásokat gyártottak, de napjainkban már okos háztartási készülékeket is készítenek. Az egyik ilyen termékük a Xiaomi Mi Robot névre hallgató okosporszívó. Ez készülék SLAM algoritmust használva térképezi fel a környezetét, így optimalizálva a takarítást pontosságát és gyorsaságát. A készülékben több processzorral van ellátva: négymagos ARM Coretex A7 és STM Microelectronics ARM Coretex M3. Telefonos applikáció segítségével a felhasználó kiválaszthatja azokat a helységeket, ahol az okosporszívó dolgozhat, ez a művelet a 10 ábrán látható. Ez a készülék ára közel 450\$. [22]



10. ábra: Okosporszívó telefonos applikációja

▪ Önvezető járművek

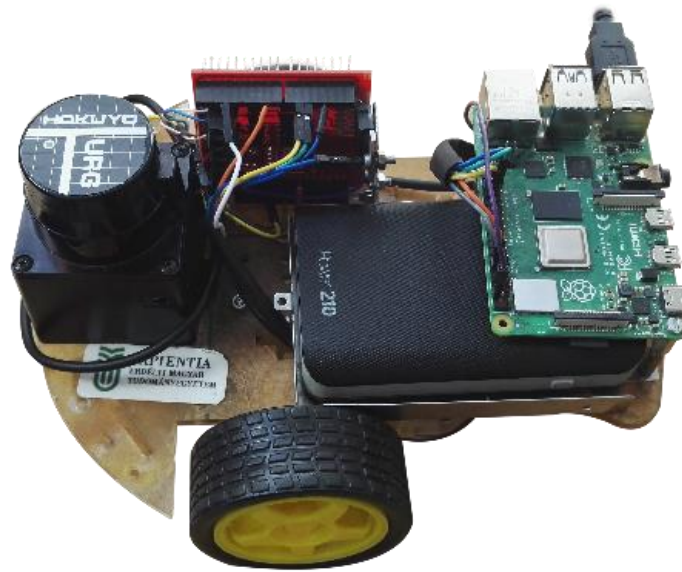
Az önvezető járművek fejlődését öt csoportba sorolhatjuk. Napjainkban az autók számos vezetési asszisztenssel vannak felszerelve, ami az önvezetés első szintjének felel meg. Például az adaptív sebességtartó automatika (ACC), amely nem csak a kívánt sebességet is tartja, hanem az előre meghatározott távolságot is előtte haladó autótól. Feladat elvégzéséhez LIDAR érzékelőket és kamerákat alkalmaznak. [23]

▪ Okostelefon

Az Iphone 12 PRO kameracsoportjába egy LIDAR érzékelő is helyet kapott. Portré készítésekor segít a testrészeket fókuszban tartani, így sokkal jobb minőségű kép készíthető. A lézeres érzékelőt fizikai távolságok mérésére is alkalmazhatjuk. [24]

3. A rendszer specifikációi és architektúrája

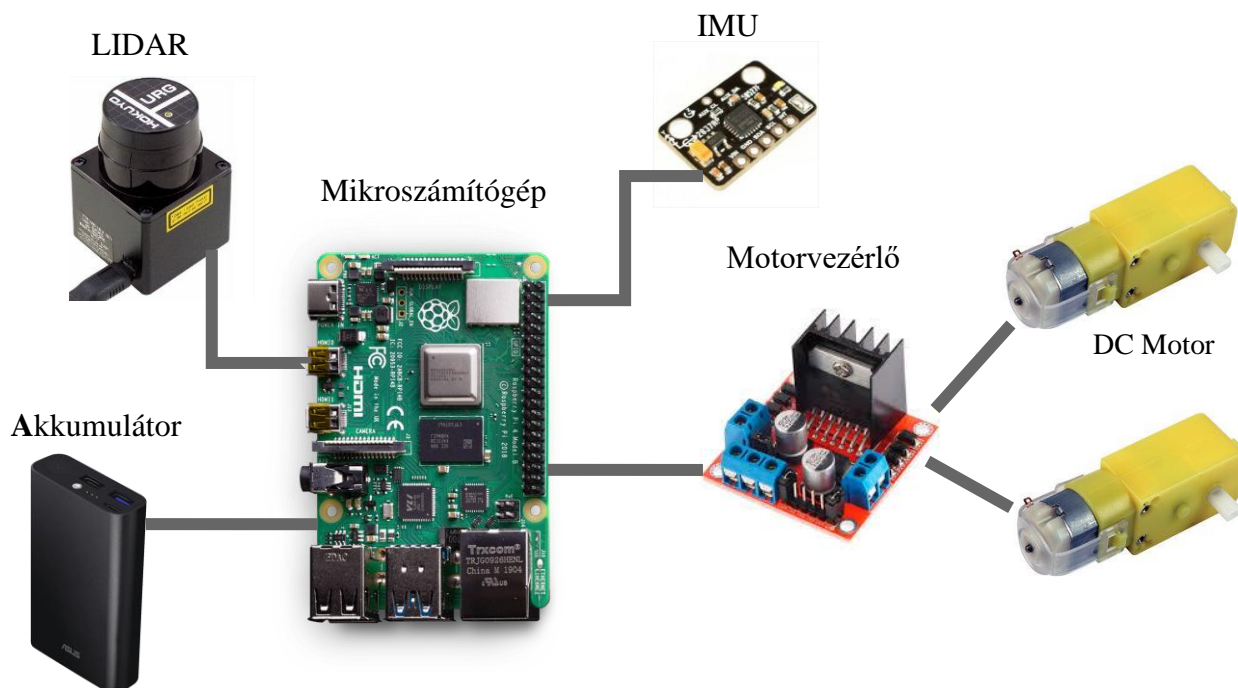
A mobilis robot két darab egymástól független 65 milliméteres átmérőjű aktív kerékkel, valamint egy görgő kerékkel rendelkezik. Így mindössze három pontban érintkezik a talajjal, így kellő stabilitással tud haladni síkfelületen. Előnye, hogy viszonylag egyszerűen irányítható, hátránya, hogy a komplexebb terepakadályokon, például lépcsőn, nem képes tovább haladni. A két aktív kerék gumifelülettel van ellátva, hogy mozgáskor kellő tapadása legyen a talajhoz. Viszonylag kis súlyú és méretű robot valósítottam meg, mérete 200 x 160 x 100 mm. A mobilis robot szerkezeti felépítése a 11. ábrán látható:



11. ábra: Megvalósított mobilis robot

3.1. Hardver felépítése

A rendszer központi vezérlőegységét egy Raspberry Pi mikroszámítógép képezi. A térképkészítés szükséges pontfelhőket egy Hokuyo URG LIDAR szenzor állítja elő, valamint a robot orientációjának meghatározására egy GY – 91 nevezetű inerciális mérő egységet (IMU) használtam. A robot mozgásért két darab 6V-os DC motort felel, amit egy L292N erősítővel vezéreltem meg. A robot energiaellátásáért egy 6000 mAh akkumulátor felel. A következő ábrán (12. ábra) látható a mobilis robot tömbvázlata.

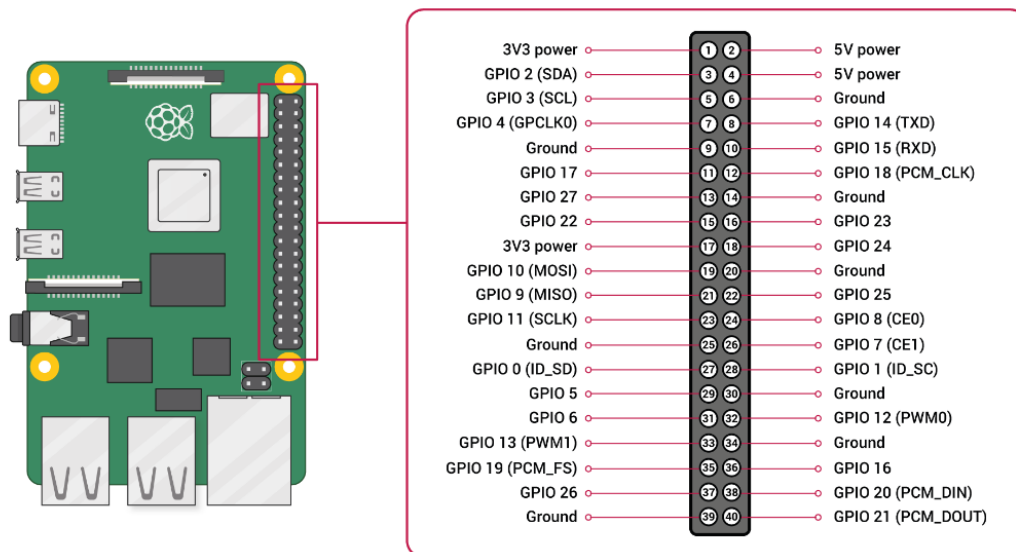


12. ábra: Mobilis robot tömbvázlata

3.1.1. Raspberry Pi 4

A Raspberry Pi 4 mikroszámítógépen összesen 40 darab GPIO láb található meg. A következő ábrán (13. ábra) lábkiosztása látható. A motor vezérléshez szükséges két darab PWM (Pulse-width modulation) jelet a GPIO 19 – 13, és 18 – 12 láb, valamint a betáplálásához szükséges feszültségeket a 4. 6. és 34. láb biztosítja. Az IMU és a Raspberry Pi közti I²C típusú kommunikáció megvalósításánál a GPIO 2 láb biztosítja az adatvonalat (SDA) és a GPIO 3 pedig az órajellett (SCL).

[12. ábra forrása] <https://www.asus.com/Accessories/Power-Banks/All-series/ZenPower-10050C-QC/>
<https://opencircuit.shop/Product/DC-Geared-Motor-1-120>
<https://arduino.blaisepascal.fr/pont-en-h-l298n/>
https://en.wikipedia.org/wiki/Raspberry_Pi#/media/File:Raspberry_Pi_4_Model_B_-_Top.jpg
<https://www.robotshop.com/en/6-dof-gyro-accelerometer-imu-mpu6050.html>



13. ábra: Raspberry Pi 4 lábkiosztása

A tárhelyet egy ADATA márkájú 16GB SD (Secure Digital) kártya képezi. Ez a flash alapú tároló 10-es osztályú besorolást kapott, szóval 10MB/sec a minimális adatátviteli sebessége. Az SD kártya formázása a Raspberry Pi Imager v1.4 program segítségével készült. Ez a szoftver a gyártó oldalán ingyenesen beszerezhető. Jelen esetben gyártó által ajánlott Raspberry Pi OS 32 bites operációsrendszert került telepítésre.

Mivel az IMU és Raspberry között I²C kommunikációs protokollt alkalmaznak, ezért szükséges volt az operációs rendszer beállításában engedélyezni ezt a kommunikációs protokollt. Az engedélyezés nélkül csak ki/bemeneti portként működik. Az érzékelő csatlakoztatása után ellenőrizni tudjuk a csatlakoztatott eszközök címét a terminálban, ezt a következő parancsok tehetjük meg: `sudo i2detect -y 1`.

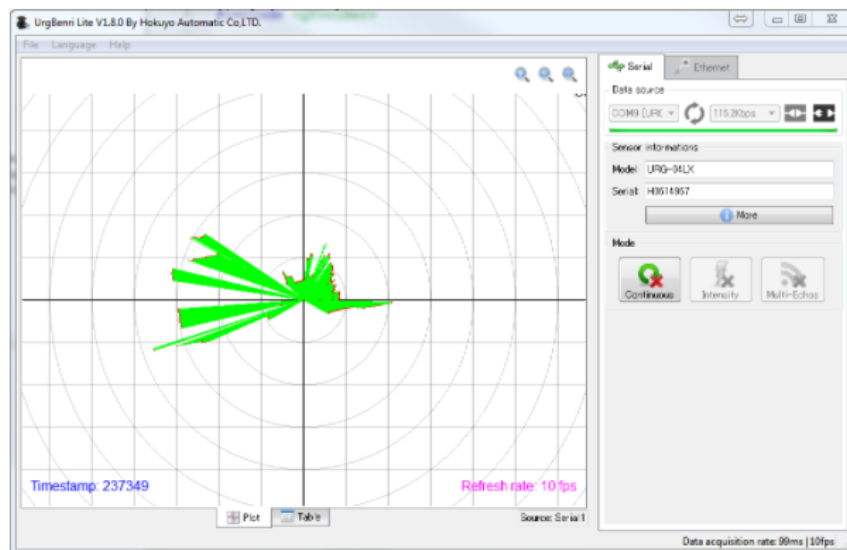
3.2. Szoftver tervezése és kivitelezése

A tervezést feladatot részfeladatok meghatározásával kezdtem. Alpontokra bontva a feladatot könnyebben átlátható, ezáltal a megvalósítása is egyszerűbb. A következőkben a LIDAR érzékelőtől való adatgyűjtést, a mobilis robot orientációjának meghatározását, irányítását és a SLAM algoritmus megvalósítását fogom bemutatni.

3.2.1. LIDAR érzékelővel való kommunikáció

Legelső lépésként a LIDAR szenzorral való kommunikációt valósítottam meg. A legelső adatbegyűjtés az UrgBenriPlus névre hallgató szoftver segítségével történt meg. Ez a szoftver a Hokuyo weboldalán található meg. Az érzékelő a számítógéppel való csatlakoztatása után,

gombnyomásra kirajzolta mért értékeket. A gyártó által biztosított szoftver a következő ábrán (14.ábra) látható.



14. ábra: UrgBenri szoftver

A nyers mérési adatok kinyeréséhez szükséges volt egy hasonló program elkészítése. A kommunikációs protokoll megismerése érdekében készítettem egy kezelőfelület ahol adatcsomagokat tudtam küldeni soros porton a LIDAR érzékelő számára. Tervezési fázist egy használati eset diagram (Use Case Diagram) szerkesztésével kezdtem, ahol leszkögeztem azokat a funkciókat és követelményeket amiket a programtól vártam el. A kezelőfelület C# nyelven íródott Visual Studio 2019 Pro fejlesztő környezetben. A programot két kódrészletre csoportosítható: a program algoritmusát tartalmazó és a felhasználói felületet megvalósító kódrészletre. A két rész logikailag összekötöttségben vannak, a felhasználói felületen történő események meghívják a feladatot elvégző függvényeket. Ezeket a függvényeket úgynevezett eseménykezelő függvényeknek nevezzük. Az érzékelőtől érkező adatokat hasonló eseményfigyelő függvények segítségével figyeltem. Megvalósítottam egy SerialConnection nevetű osztályt, amely segítségével soros porton tudtam csatlakozni az érzékelőhöz. A csatlakozás megkönnyítése érdekében, kapcsolódáskor lekérdeztem a csatlakoztatott eszközök portját (COM) és csatlakoztam egy az első porthoz. Így minden indításkor nem kellett beállítani a COM portot, egy plusz művelettől szabadul meg a felhasználó. Sikeres csatlakozás jelzésére, lekérdeztem az érzékelő paramétereit és kiíratam az interfészre. Az előző fejezetben bemutatott MD parancs segítségével olvastam be a mért távolságokat. Az érzékelőtől kapott választ két karakteres kodolással határoztam meg, ezt az 1. kódrészlet segítségével valósítottam meg.

```
// === Decode ===
private static long decode(string data, int size, int offset = 0)
{
    long value = 0;
    for(int i = 0; i < size; i++)
    {
        value << 6;
        char debug = data[offset + i];
        value |= (long) data[offset + i] - 0x38;
    }
    return value
}
```

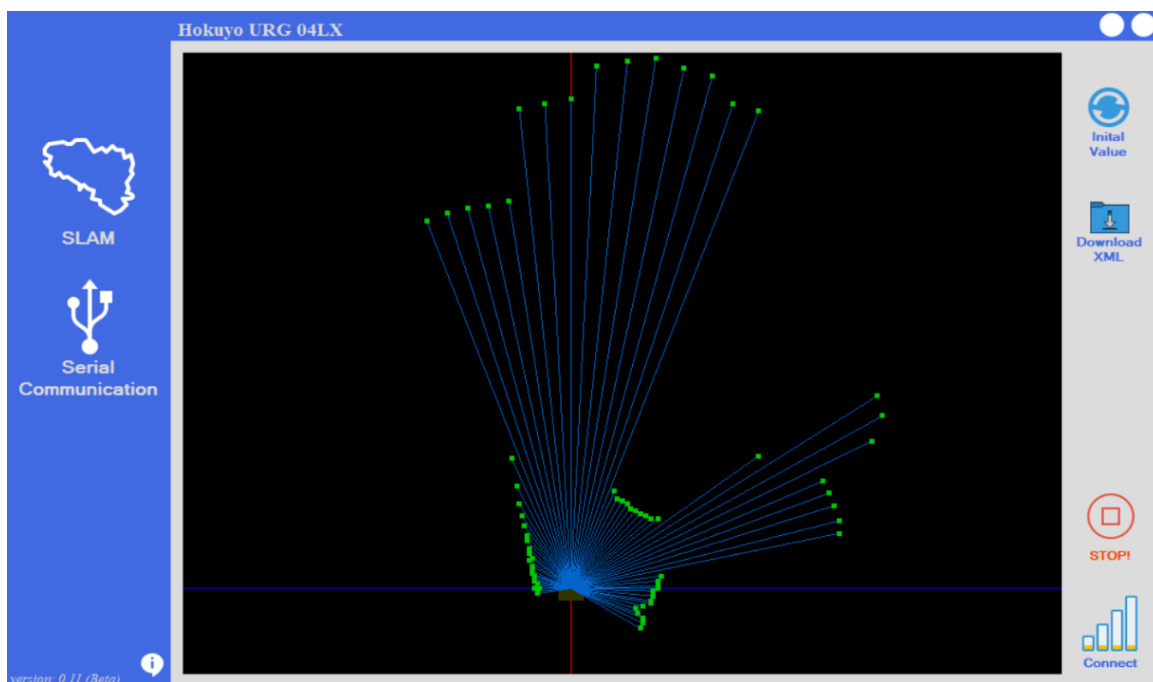
1. kódrészlet: LIDAR mérések dekódolása

A mérési eredmények ábrázolását egy vászonra (canvas) készítettem el. A vászon felépítése hasonlít kétdimenziós mátrixéhoz. Például egy pont ábrázoláshoz szükséges egy X és Y koordináta, ez egy mátrix oszlop és sor számához hasonlítható. Egy mért adat megjelenítéséhez szükséges volt két koordináta meghatározásához: Egy pontra, ahonnan a vonal kezdődik és egy másik pontra, amit a vonal végét jelent. Ezeket az egyeneseket egy kör mentén kell ábrázolni ezért az X és Y koordinátákat a következő képpen számoltam ki (1. egyenlet):

$$\begin{cases} X = \text{távolság} * \sin \frac{\pi * \text{szög}}{180} \\ Y = -\text{távolság} * \sin \frac{\pi * \text{szög}}{180} \end{cases}$$

1. egyenlet: X és Y pont meghatározása

A méréseket egy bizonyos érték közé kellett normalizálnom, hogy a vászonra a legtávolabbi pont is felférjen. A 15. ábrán a mért értékeket kirajzolása látható. Zöld ponttal jelöltem az akadályok távolságát.



15. ábra: LIDAR mérések kirajzolása

A LIDAR érzékelő kommunikációs protokoll megismerése és a pontfelhők sikeres ábrázolása után, a robot vezérlőegységre, a Raspberry Pi-re is implementáltam ezt a kódot.

3.2.2. IMU mérés és kalibrálás

A Raspberry Pi és az MPU9250 IMU érzékelő között I²C kommunikáció protokollt alkalmaztam. A mikroszámítógép GPIO 2 - es lábát használom adatvonalnak (SDA) és a GPIO 3 – as lábát pedig az órajelnek (SCL). Az érzékelő betápláláshoz szükséges feszültséget (3.3 Volt) és földet (GND) a Raspberry 1 és 8 lábairól kapja meg. Jelen esetben a mobilis robot vezérlőegysége nem kommunikál a barométerrel (BMP280), mivel nincs szükségünk légnyomás mérésekre.

Az érzékelő regisztereinek beállításához és kommunikációs protokoll megvalósításához egy Python nyelven készült FaBo9Axis nevű könyvtárát használtam fel [25]. A könyvtárban lévő függvények csak egy részét használtam fel, többek között az MPU9250 és az AK8963 modul inicializálását és giroszkóp, gyorsulás, magnetométer beolvasását végző függvények. Az MPU9250 modul inicializálásánál az érzékelőket a következőkre skáláztam: A giroszkópot ± 250 dps - re és a gyorsulásmérőt $\pm 2g$ – re. Az AK8963 modul esetében, a magnetométer kimeneteinek méretét 16 bitesre állítottam. Az smbus [26] nevezetű könyvtár segítségével építi fel a kommunikációt. Ez a könyvtár a Raspberry Pi I²C buszát képviseli. A buszon küldött és fogadott adatok az SMBus objektumon keresztül történik. Ezáltal könnyen implementálható ez a

kommunikáció. A buszra való írás a *write_i2c_block_data* (*I²C_cím*, *regiszter*, *adata*) és az olvasás a *read_i2c_block_data* (*I²C_cím*, *regiszter*, *hossza*) függvények segítségével történik.

3.2.2.1 Magnetométer modell

A robot szögváltozását mérő giroszkópérzékelők a szögsebesség idő szerinti integrálja a mért szögelfordulást adja. A giroszkópok a nehézségi erőtől függetlenül követik a relatív elmozdulást, ezért az érzékelők hibájából helyzeteltérések jelentkezhetnek, amit „elsodródásnak” (drift) nevezzünk. A magnetométerek a mágneses térerőt mérik, így képesek meghatározni a Föld mágneses mezőjének szögét, valamint a gyorsulással összehasonlítva nagy pontossággal meghatározható a mobilis robot iránya. A mágneses érzékelőknél megjelenhetnek kemény és lágyvas torzítások. A keményvas probléma akkor merül fel, amikor az érzékelő közelében állandó mágnesek jelennek meg. Lágyvas torzításnak nevezzük, ha egy test a mágneses teret torzítja, de nem generál mágneses teret, például a vas vagy nikkell jelenléte. A magnetométer hibái szűrésére egy Riki nevezetű National University of Singapore egyetemen végzett tanulmányát használtam fel [27], amit a következőkben szeretnék bemutatni:

Szükséges a bemenetek skálázása, hogy a kimenetek egy kívánt tartományba essenek. Ez matematikailag egy diagonális mátrixszal érhető el (2. egyenlet):

$$S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix}$$

2. egyenlet: Bemenetek skálázása

Az érzékelő beállításkor a tengelyek valószínűleg kissé el lesznek tolva, mivel nem tudjuk tökéletesen pozícionálni. Ez azt eredményezheti, hogy a mérési tengely ferde lesz (3. egyenlet). A mátrix oszlopai az ortogonális tengelyhez viszonyított elfordulást jelölik.

$$N = \begin{bmatrix} n_{x,x} & n_{y,x} & n_{z,x} \\ n_{x,y} & n_{y,y} & n_{z,y} \\ n_{x,z} & n_{y,z} & n_{z,z} \end{bmatrix}$$

3. egyenlet: Tengely kompenzálás

A torzítási hiba, azt jelenti, hogy egy konkrét értékkel eltolja a mérésünket. Ezt egy egyszerű eltolási vektorral modellezhetjük. (4. egyenlet)

$$b_m = [b_{m,x} \ b_{m,y} \ b_{m,z}]^T$$

4. egyenlet: Torzítási hiba

A keményvas torzítás hasonló az előbbi modellhez, ezt b_{hi} jelöljük. Míg a keményvas torzítás tájolástól függően állandó, addig a lágyvas torzítás az érzékelő és a torzítótest közelségétől függ. Így egy bonyolultabb mátrixszal modellezhetjük, ahol az oszlopok az adott tengely tájolását és méretarányát jelölik az ortogonális tengelyhez képest (5. egyenlet).

$$A_{Si} = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix}$$

5. egyenlet: Lágyvas torzítás

A fenti hibák matematikai modelljét felhasználva, felírhatjuk a magnetométer modelljét. A h az aktuális mágneses mezőt és a h_m a magnetométer hibás mérését jelenti. (6. egyenlet)

$$h_m = A \cdot h + b, \quad ahol \begin{cases} A = SNA_{Si} \\ b = SNb_{hi} + b_n \end{cases}$$

6. egyenlet: Magnetométer modellje

3.2.2.2 Magnetométer kalibráció

A 6. egyenletből kifejezhetjük a h -t, akkor a következő egyenletet kapjuk. (7. egyenlet)

$$h = A^{-1}(h_m - b)$$

7. egyenlet: magnetométer modellje kifejezve

Ismerve az A inverz és b mátrix értékét meghatározható a tényleges mágneses mező értéke. A következőkben ezeket az mátrixokat fogjuk megbecsülni. Mivel a mágneses mező nagyságát 1-re akarjuk skálázni, ezért a 8. egyenlettel írhatjuk fel:

$$h^T h = 1$$

8. egyenlet: Mágneses mező

A 8. számú egyenletet, ha behelyettesítjük a 7. egyenletbe, akkor a 9. egyenletet kapjuk.

$$(h_m - b)^T (A^{-1})^T A^{-1} (h_m - b) = 1$$

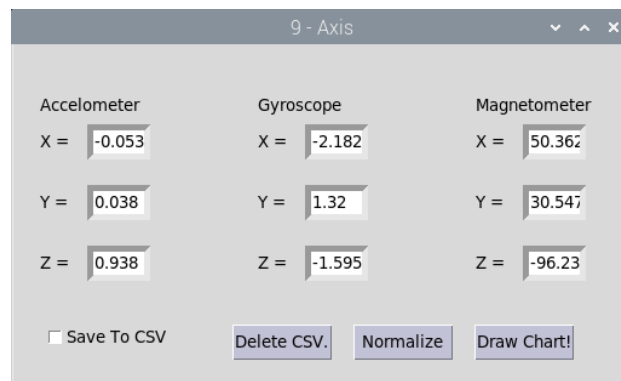
9. egyenlet: Behelyettesítve a magnetométer modelljébe

Legyen a $Q = (A^{-1})^T A^{-1}$. Mivel a Q mátrix szimmetrikus, ezért a fenti egyenletet egyszerűsíthető és felírható kvadratikusan. (11. egyenlet)

$$h_m^T Q h_m + h_m^T n + d = 0, \quad ahol \begin{cases} n = -2Qb \\ d = b^T Q b - 1 \end{cases}$$

10. egyenlet: Kvadratikus formában

Feltételezhetően a h az egy egységnyi irányvektor, így az összes mérés egy egységnyi kört kellene alkosson. Tehát a 11. egyenlet egy elliptikus felület (A kör az ellipszis sajátos esete) egyenletének kell lennie. A kalibráció elvégzéséhez először a nyers adatokat kellett beolvasnom és eltárolnom egy excel táblázatban. Az értékeket egy interfész segítségével megjelenítettem, így az esetleges kommunikációs hibákat kiszűrhettem. A felhasználói felület Python nyelven készült Tkinter állomány felhasználásával, a grafikus ábrázoláshoz pedig a Matplotlib állományt implementáltam. A 16. ábrán látható a felhasználói felület.



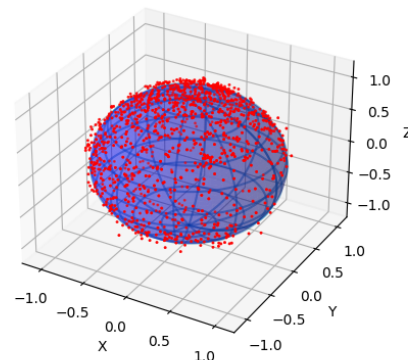
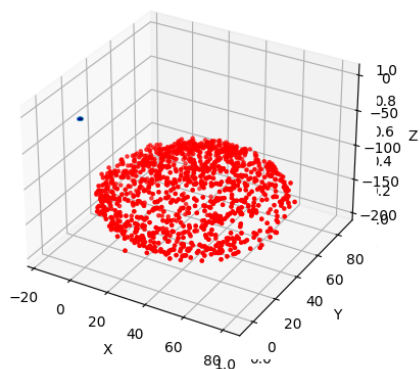
16. ábra: IMU mérések

Az A^{-1} és b mátrix kiszámításához felhasználtam a fent említett [27] példakódját. A mátrixok a következő képen lettek kiszámolva. (2. kódrészlet)

```
Qinv = np.linalg.inv(Q)
b = -np.dot(Qinv, n)
Ainv=np.real(1/np.sqrt(np.dot(n.T, np.dot(Qinv,n))-d)*linalg.sqrtm(Q))
```

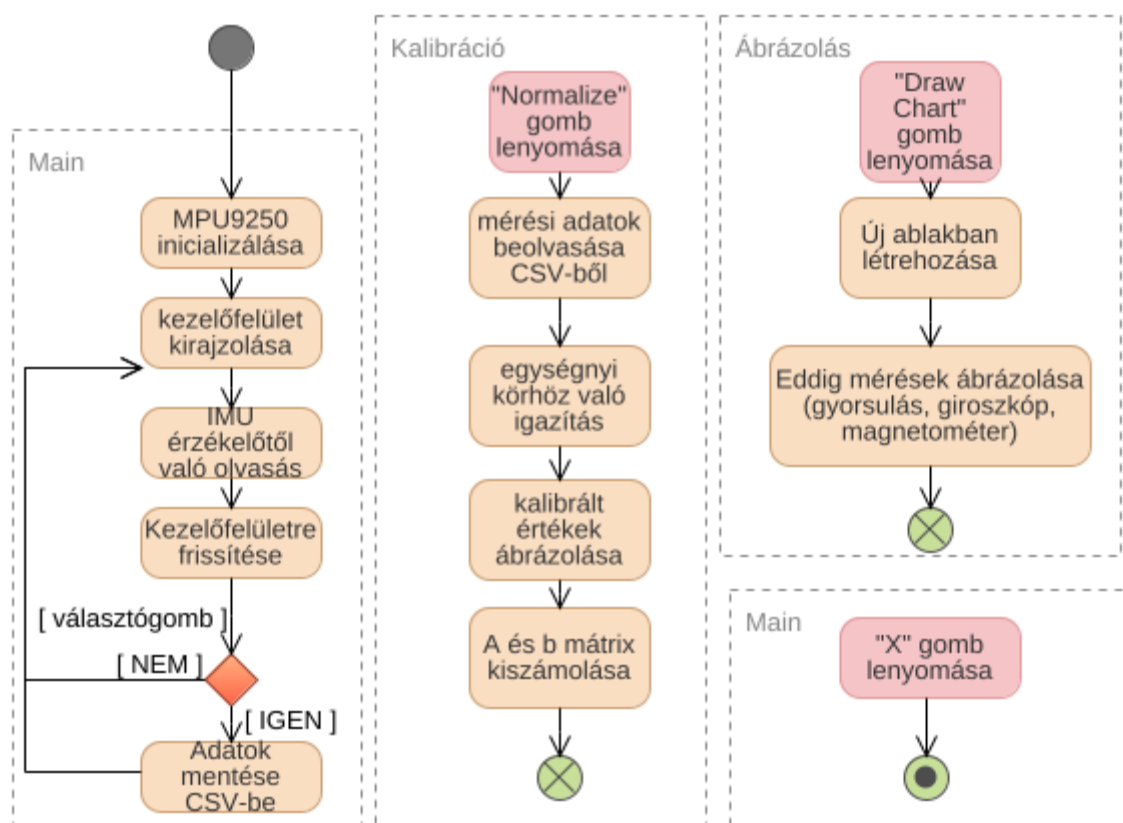
2. kódrészlet: A és b mátrix számolása

A 17. ábrán látható a mért magnetométer értékek kalibráció előtt, valamint után. Pirossal látható a mért értékek és kekkel az egységsugarú kör. Jól megfigyelhető, hogy kalibráció után, a magnetométer mérések az egységsugarú körön belül helyezkednek el.



17. ábra: IMU kalibráció

A program működését modellező aktivitás diagramot a [app.genmymodel](http://app.genmymodel.com) [28] online platformon készítettem el. Az aktivitás diagram segítségével a program tevékenységeit és elvégzett műveleteit ábrázolhatjuk. A 18. ábrán látható az IMU értékek beolvasó és kalibráló szoftver aktivitás diagramja. A gombok lenyomásával egy eseményt váltunk ki, ezeket az eseményeket feldolgozó függvényeket piros téglalappal jelöltem. Az ábrán a folyamat kezdetét egy fekete kör jelzi és a folyamat végét pedig egy fekete körrel, zöld körvonallal.



18. ábra: IMU szoftver aktivitás diagramja

3.2.2.3 Magnetométer szűrése

A mérések alapján a magnetométer mérések zajosak voltak. Ezért megvalósítottam egy Alfa szűrőt. Az alfa szűrő karakterisztikája miatt lassúnak bizonyult, az orientáció változásokat nem tudta elég gyorsan lekövetni, így alkalmaztam a végtelen impulzus válasz (IIR) szűrőt, ami a mérések során jóval gyorsabbnak bizonyult.

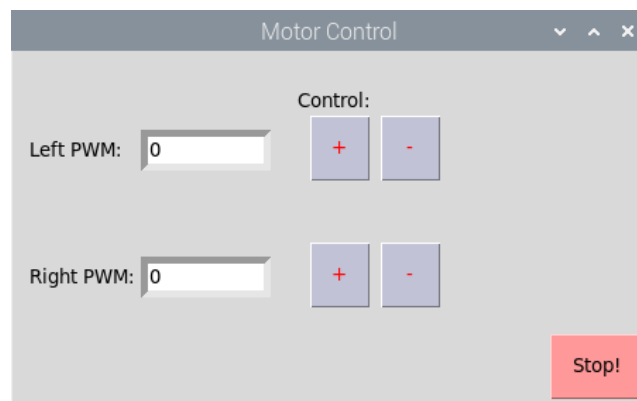
A magnetométer szűrésére egy másodfokú aluláteresztő IIR szűrőt alkalmaztam. A szűrő megvalósításához signal [29] nevezetű Python könyvtárat használtam fel. Ebben a könyvtárban számos más szűrőt is megtalálhatunk leimplementálva. A *signal.ellip* függvény használtam az elliptikus szűrőhöz, paramétereként megkellett adnom a szűrő fokát (N), az egységnyi erősítés alatt megengedett maximális hullámmást az átviteli sávban (rp), a minimális csillapítást a leállósávban (rs), a kritikus frekvenciát, vagyis azt a pontot, ahol az erősítés először csökken átviteli sáv alá (wn) és a szűrő típusát. A szűrő a következő képen volt implementálva: (3 kódrészlet)

```
sos = signal.ellip(2, 1, 100, 0.2, 'lowpass', output = 'sos') #lowpass
filteredTheta = signal.sosfilt(sos, data)
```

3. kódrészlet: IIR szűrő megvalósítása Python nyelven

3.2.3. Motor szabályozása

Az L292N típusú motorvezérlő impulzusszélességű jelek (PWM) segítségével vezérli meg motorokat. Összesen négy darab PWM jel volt szükséges a motorok mozgatására. Elsősorban a egy kezelőfelület készítettem, ahol külön lehet szabályozni a motorok forgási sebességét, valamint irányát. Ez kis program azért volt hasznos, mert a szabályzóhoz szükséges paramétereket így határoztam meg és így ellenőrizni is tudtam, hogy motorvezérlő számára helyesen küldöm ki a PWM jeleket. A következő ábrán (19. ábra) program kezelőfelülete látható.

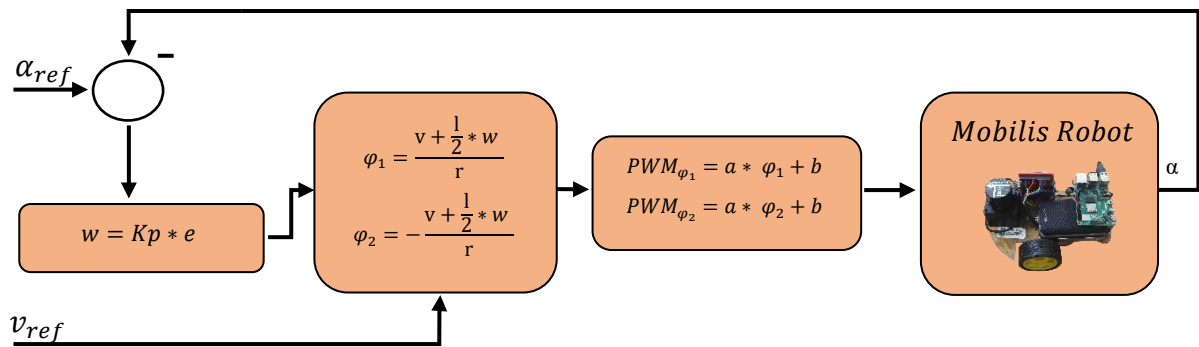


19. ábra: Motor vezérlő szoftver kezelőfelülete

3.2.3.1 Szög mentén való vezérlés

A mobilis robot koordinált mozgatására szükségem volt egy olyan szabályozóra, amely egy előírt szög mentén szabályozza a robotot. A vezérlőjelek meghatározásához szükségem volt a robot kerék átmérőre, a tengelyhosszra és a, b paraméterre, amit kísérleti úton határoztam meg. A robot szerkezeti felépítését megmérve, a kerék átmérő értékét 5.5 – re és a tengelyhossz méretét pedig 11.1 – re állítottam. A 20. ábrán látható a szabályzó algoritmus.

- l – tengelyhossz
- r – kerék sugara
- v – robot lineáris sebessége
- w - robot szögsebessége
- α - robot elfordulása
- Kp - erősítés



20. ábra: Mobilis robot szabályozási diagramja

A fenti ábrán látható diagramot a motorvezérlő osztályban implementáltam. A szabályzó függvénynek összesen három paraméterre van szüksége: a referencia szög, a mért szög és az előírt szögsebesség. Python nyelven megvalósított szabályozó látható a 4. kódrészleten.

```
# === Robot Control ===
# parameters: ref angle, measured angle, ref angular velocity
# return: 2 PWM singal -> leftPWM, rightPWM
def robotControl(self, alfaRef, alfa, Vref):
    angularSpeedRobot = self.Kp*(alfaRef-alfa)
    linearSpeedRobot = Vref

    leftWheelAngularSpeed = linearSpeedRobot +/
self.half_axis*angularSpeedRobot)/self.wheelRadius
    rightWheelAngularSpeed = linearSpeedRobot -/
self.half_axis*angularSpeedRobot) / self.wheelRadius

    leftMotorPWM = self.a * leftWheelAngularSpeed + self.b
    rightMotorPWM = self.a * rightWheelAngularSpeed + self.b
```

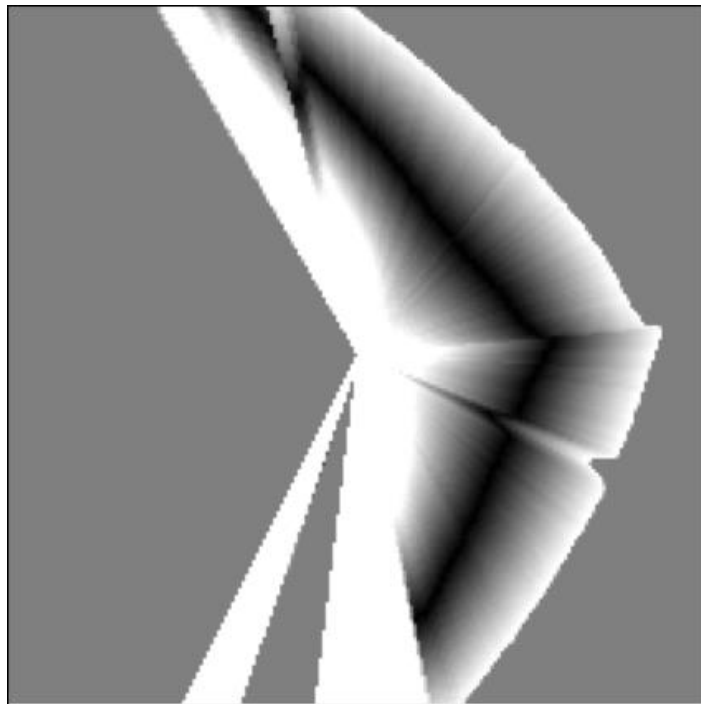
4. kódrészlet: Motor szabályzó Python nyelven

3.2.4. SLAM algoritmus

SLAM algoritmus megvalósításához egy Breezy SLAM [30] nevezetű nyílt forráskódú állományt használtam fel. Ez a könyvtár CoreSlam nevezetű algoritmus segítségével készíti el a térképet. A fejlesztő minél egyszerűbb megvalósítást tűzte ki célul, ezért ez az algoritmus jóval egyszerűbb megoldás biztosít a DP-SLAM (Distributed particle - SLAM) el szemben. A DP-SLAM előnye, hogy a hosszú folyósokon, valamint nagy terekben sem veszti el a robot pozícióját. A CoreSLAM két fő függvénnyel rendelkezik:

A *ts_distance_scan_to_map*, amely valószínűségi függvényként működik és a mért, helyzeti állapotok tesztelését végzi el. Ez egy nagyon gyors eljárás, mivel csak egész számokkal végez egyszerű összeadást.

A *ts_map_update* függvény a robot mozgása közben felel a térképkészítéshez. Az algoritmus az észlelt akadályokhoz nem egyetlen pontot rajzol, hanem egy “lyukkal” ábrázolja. Tehát a térképen található lyukak mélysége jelenti a tényleges akadály pozícióját. A térkép méretét lecsökkentve jól demonstrálható, hogy hogyan rajzolódik ki egy akadály. A 21. ábrán átható, hogy egy akadályt valós helyzetét a “legmélyebb”, vagyis a legsötétebb pont jelzi.



21. ábra: Térkép szerkezete

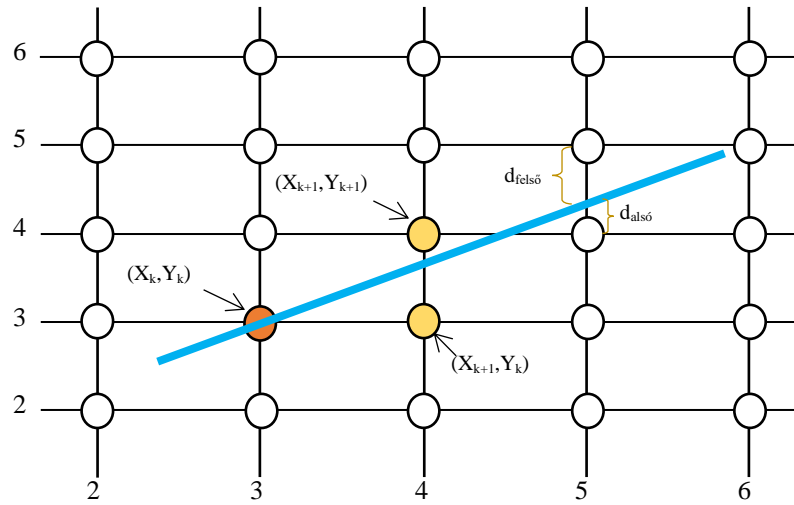
A térkép vázát egy kétdimenziós byte tömb képezi. Minden tömb elem egy pixelt jelent a térképen, minél nagyobb tömbbel dolgozunk annál részletesebb térképet kaphatunk, viszont annál nagyobb számítás kapacitás szükséges az előállításához, valamint a megjelenítéshez. A térképén inicializálásakor szürkére állítjuk. A tömb elemei értéke határozzák meg a kirajzolt térkép színét, vagyis ahol fehér pixelt látunk, ott a tömb eleme 0x00 és ahol fekete pixelt ott 0xff.

A térképbe való integrálás egy alfa-beta szűrőn keresztül történik. Ennek a szűrőnek az előnye, hogy nem igényel részletes rendszermodellt, a szűrő feltételezi, hogy a rendszert megfelelően közelíthető. Két állapottal rendelkezik, ahol az első állapotot a második állapot időbeli integrálásával kapjuk. A *ts_map_update* egy Bresenham-algoritmust is használ a LIDAR sugarak térképre való rajzolásához.

3.2.4.1 Bresenham algoritmus

A Bresenham algoritmus [31] egy olyan vonalrajzoló algoritmus, amely kiválasztja egy n – dimenziós pontmátrix azon elemeit, amelyek szükséges egy egyenes ábrázolásához. Általában ezt az algoritmus bit térképek rajzolásához használják, mivel csak egész számokkal dolgozik. Ebből kifolyólag gyors algoritmusnak számít.

Egy egyenes meredeksége a következő egyenlettel írható le: $f(x) = y = mx + b$, ahol az m a meredekséget és a b az y metszéspontját jelöli. A meredekség felírható a következő formában is: $m = \frac{dy}{dx}$. A 22. ábrán egy példát szeretnék bemutatni. A 3,3 számú pozícióból kell eldönteni, hogy melyik a következő legközelebbi pont az egyenestől (4,4 és a 4,3 pont közül kell választani).



22. ábra: Bresenham algoritmus példa

Ehhez a függőleges távolságokat az egyenestől, $d_{alsó}$ és $d_{felső}$ jelölhetjük, amit a 11. egyenlet segítségével tudunk meghatározni:

$$\begin{cases} d_{alsó} = y - y_k = m(X_k + 1) + b - Y_k \\ d_{felső} = (y_k + 1) - y = Y_k + 1 - m(X_k + 1) - b \end{cases}$$

11. egyenlet: távolság számolás

Egyszerű különbség számolással könnyen eldönthetjük, hogy melyik pixel van közelebb az egyeneshez. (12. egyenlet)

$$d_{alsó} - d_{felső} = 2m(x_k + 1) - 2y_k + 2b - 1$$

12. egyenlet: közelebbi pont kiszámolása

Az így kapott képletbe az m -et a következővel helyettesítjük $m = \frac{dy}{dx}$, majd egyszerűsítés után a 13. egyenletet kapjuk.

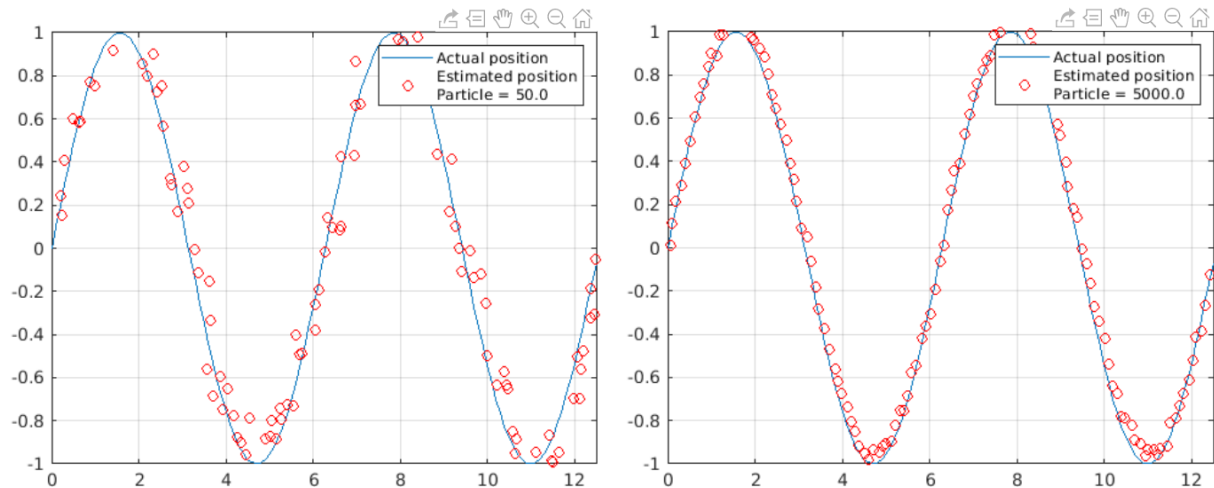
$$d_x(d_{alsó} - d_{felső}) = 2d_yx_k - 2d_xy_k + 2dy + 2d_x(2b - 1)$$

13. egyenlet: közelebbi pont kiszámolása2

A különbség negatív, akkor az alsó pixelt választjuk, ellenkező esetben pedig a felső pixelt.

3.2.4.2 Részecskeszűrő

A részecskeszűrő [32] a jelfeldolgozás és a Bayes – statisztika [33] számolás során felmerülő szűrési problémákat kiküszöbölésére alkalmazható. A szűrési probléma az állapotok becsléséből adódik, mivel véletlenszerű perturbáció van jelen az érzékelők és a dinamikus rendszerben egyaránt. A célja valamilyen sztochasztikus (véletlenszerű) modell állapot eloszlásának meghatározása, zajos és részleges megfigyelés által. A részecskeszűrő részecskéket, másszóval mintákat használ a sztochasztikus folyamat (véletlenszerű folyamat) utólagos eloszlásának reprezentálására. Az eloszlásból vett mintákat részecskék halmaza reprezentálja. Minden részecskéhez egy valószínűségi súlyt rendelnek. A részecskeszűrő algoritmus rekurzív módon számolja ki az állapotbecslést. Első lépésként az előző állapotok segítségével becsüli meg aktuális állapotot. Második lépésként az érzékelőktől származó adatok segítségével az előző állapotbecslést korrigálja. Majd a részecskéket újra elosztja, hogy megfeleljen az utolsó becsült állapot eloszlásának. A 23. ábrán Matlab környezetben megtalálható részecskeszűrő példaprogramot [34] próbáltam ki. Megfigyelhető, hogy több részecskét használva, pontosabb megközelítést kaphatunk.

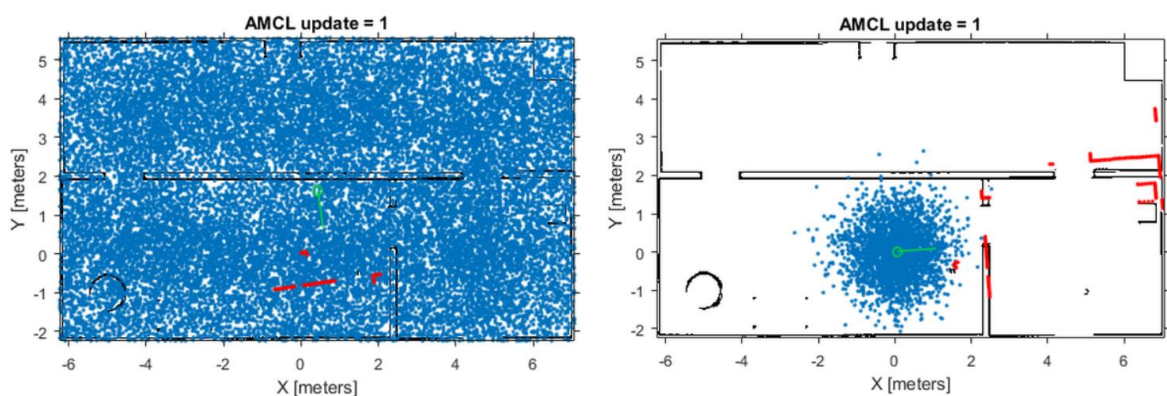


23. ábra: Matlab példa részecskeszűrőre

3.2.4.3 Monte – Carlo lokalizáció algoritmus

Az aktuális méréseket a térképre való integrálásáért és a robot pozíciójának meghatározásáért egy egyszerű Monte Carlo algoritmus [35] felel. A Monte Carlo – lokalizáció

részecskeszűrőn alapszik és mobilis robot lokalizáció meghatározására alkalmazható. A mozgó robot helyzetét a folyamatosan frissülő környezeti térkép alapján becsüli meg. Mivel részecskeszűrőn alapszik, ezért minden lehetséges pozíció egy részecskét képvisel. Általában az algoritmus kezdetén az részecskék véletlen eloszlásúak, vagyis a robot nem tudja pozícióját a térben. Minden elmozdulás után eltolja a részecskéket és megbecsüli az újabb lehetséges pozíciókat. Ezt az 24. ábrán szeretném demonstrálni. Kék színnel vannak ábrázolva a robot lehetséges pozíciói (részecskék) és pirossal színnel a LIDAR mérések. Jól látható, hogy az első periódusban véletlenszerű eloszlásban vannak a részecskék. A következő periódusban pedig egy újabb mérés, segítségével leszűkült a lehetséges pozíciók száma.



24. ábra: Monte Carlo lokalizáció

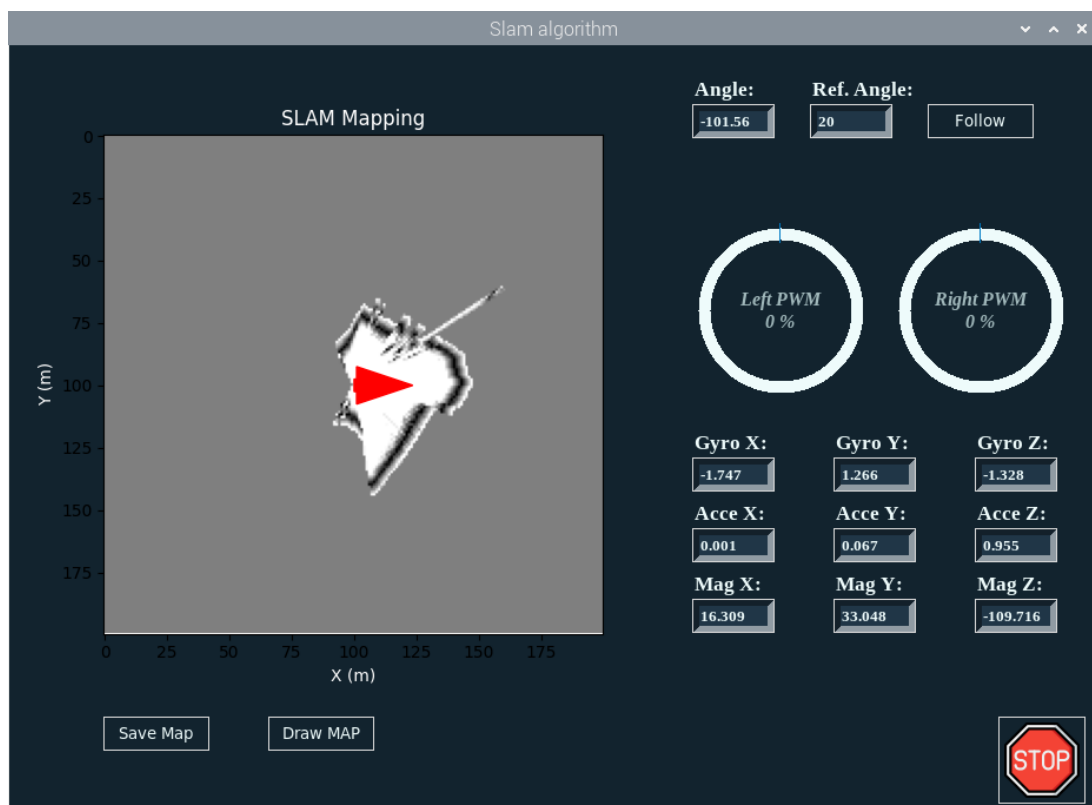
A részecskék a következő képen frissülnek:

- A részecskék egy megadott mozgásmodell szerint változnak (A mozgásmodellben a robot mozgási képességei vannak meghatározva).
- A részecskékhez valószínűségi súlyokat rendelünk a LIDAR mérések alapján.
- A súlyok alapján a megbecsüli a robot pozícióját.
- Végül a részecskéket újra elosztja, valamint a teljesítmény növelés érdekében a részecskék számát csökkenti.

3.2.5. Mobilis robot szoftvere

Megvalósítottam egy olyan kezelőfeletet, amely az előző alpontokban bemutatott alprogramokat fogja össze. A felhasználói felület megvalósítására azért volt szükséges, mert így a beolvasott értékek, a szűrt értékek és a beavatkozó jelek valós időben megfigyelhetők voltak. Valamint a felhasználói felületen megtalálható gombokhoz különböző funkciót lehetett rendelni. Megjelenítésre kerültek a IMU mérések, a szűrt magnetométeres orientáció, a kiküldött PWM szabályzójelek a motorvezérlő számára, valamint a SLAM algoritmus által készített hatalmas byte

tömb, vagyis a térkép. Az egyik legfontosabb gomb a “STOP” gomb, melynek segítségével bármikor megállíthatjuk a robot, így megelőzve azt, hogy kárt tegyen valamiben vagy önmagában. Ezen kívül még három gomb található meg: a „Follow” gombbal indíthatjuk el a vonalkövető algoritmus, a „Draw MAP” gombbal a térképrajzolást kapcsolhatjuk ki vagy be és a „Save Map” gombbal pedig a szoftver főkönyvtárba menthetjük le a térképet. A 29.ábrán a felhasználói felület látható térképkészítés közben.



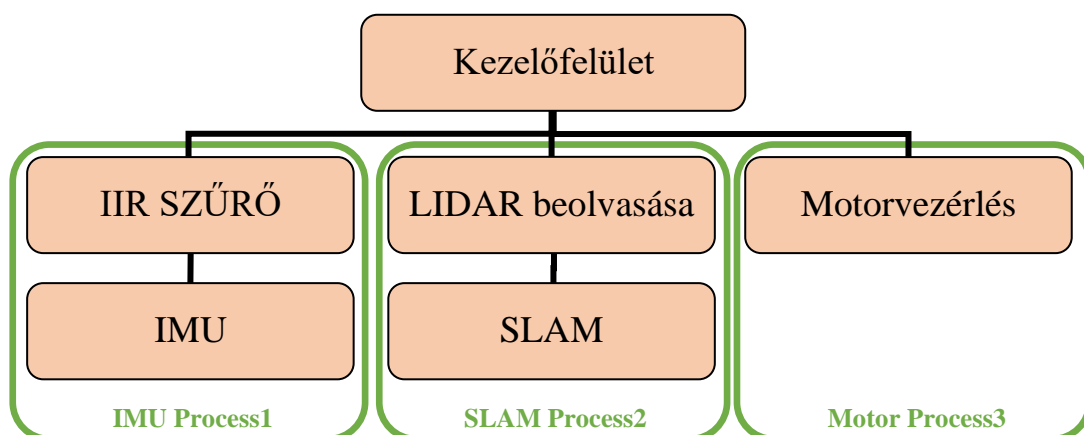
25. ábra: Mobilis robot kezelőfelülete

Mivel a mobilis robot egyidőben több időkritikus feladatot kell elvégeznie, ezért szükséges volt olyan módszert alkalmaznom, amivel párhuzamosan tudom feldolgozni a bejövő adatokat. Erre megoldást nyújt a többszálas (multithreading) vagy többprocesszoros (multiprocessing) adatfeldolgozás. A többprocesszoros és a többszálú feldolgozás közötti különbség az, hogy a többprocesszor lehetővé teszi, hogy a rendszer több folyamatot adjon hozzá a rendszerhez, míg a többszálú feldolgozás lehetővé teszi, hogy a folyamat több szálát hozzon létre a rendszer számítási sebességének növeléséhez. Vagyis a többprocesszoros rendszer egyszerre több folyamatot képes végrehajtani, míg a többszálas rendszer a több szálának egyidejű végrehajtását tudja kezelni egy folyamatban. A következő táblázatban (1. táblázat) összehasonlításra kerül a többprocesszoros és a többszálas adatfeldolgozás. [36]

Többprocesszoros	Többszálás
A többprocesszoros műveletek segítenek növelni a számítási teljesítményt.	A többszálás kezelés segít egyetlen folyamat számainak létrehozásában a számítási teljesítmény növelése érdekében.
A folyamatokat egyidejűleg hajtja végre	Egy folyamat több szálát egyidejűleg hajra végre
A folyamat létrehozása lassú és erőforrás specifikus	A szál létrehozása gazdaságos időben és erőforrásban
kevesebb idő szükséges a munka feldolgozására	Mérsékelt időt vesz igénybe a munka feldolgozására

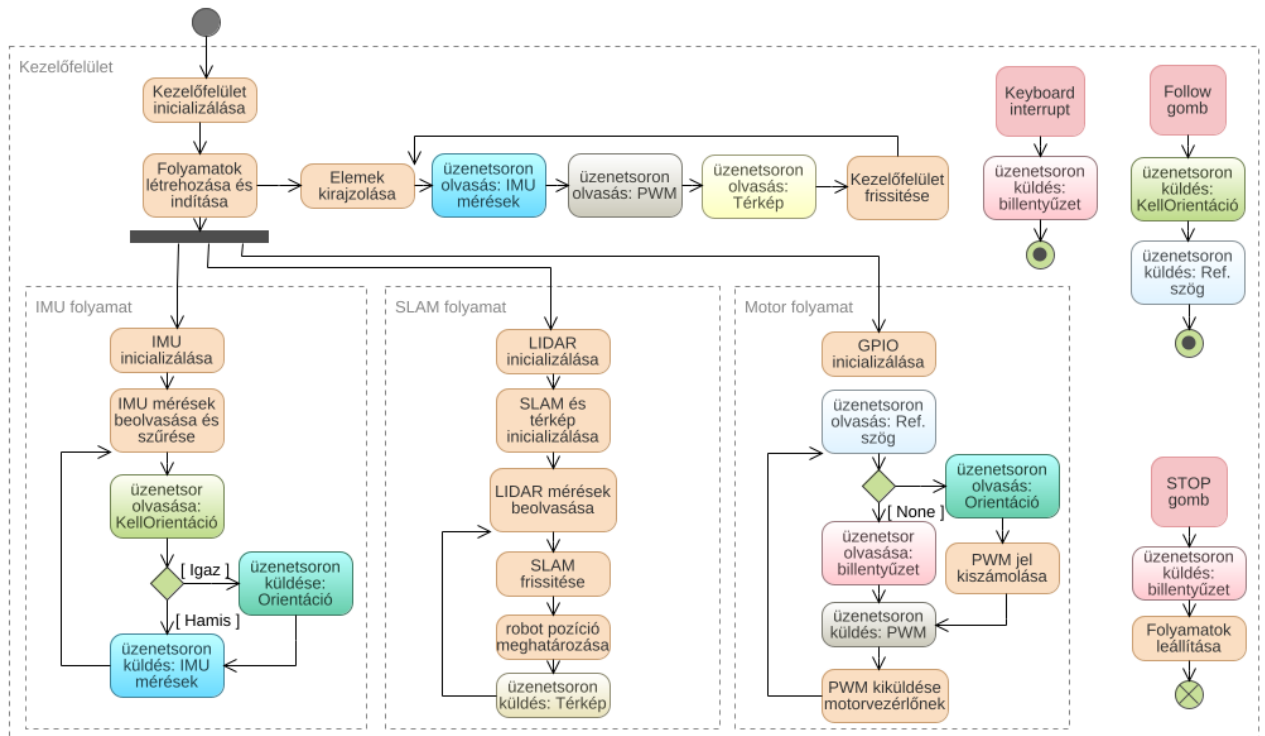
1. táblázat: Többprocesszoros és többszálás adatfeldolgozás

A többprocesszoros megoldást választottam, így a robot feladatait három nagy részre bontottam le. Az egyik folyamat a SLAM algoritmus és a LIDAR érzékelővel való kommunikációért felel, a második a folyamat az IMU beolvasásáért és ennek szűrésért, a harmadik pedig a motorvezérlésért. A folyamatok közti kommunikációt üzenetsorokkal (queue) valósítottam meg. A queue.py [37] könyvtárat használtam fel az üzenetsorok megvalósításához. Üzenetsorok létrehozásakor méretüket 10 - re állítottam, ezért hibátlan működés estén soha nem telhetnek meg, mivel fogyasztók gyakrabban ütemezőknek, mint a termelők, elkerülve azt, hogy az üzenetsorban felhalmozódjon az adat. Az adatokat a `queue.put` parancs segítségével helyeztem az üzenetsorba, paramétereit segítségével blokolásos várakozásra állítva. A `queue.get` parancs segítségével olvastam az üzenetsorból, nem blokolásos várakozásra állítva. A következő ábrán (26. ábra) látható, hogy a három folyamatot a kezelőfelület ütemezi és fogja össze.



26. ábra: Folyamatábra

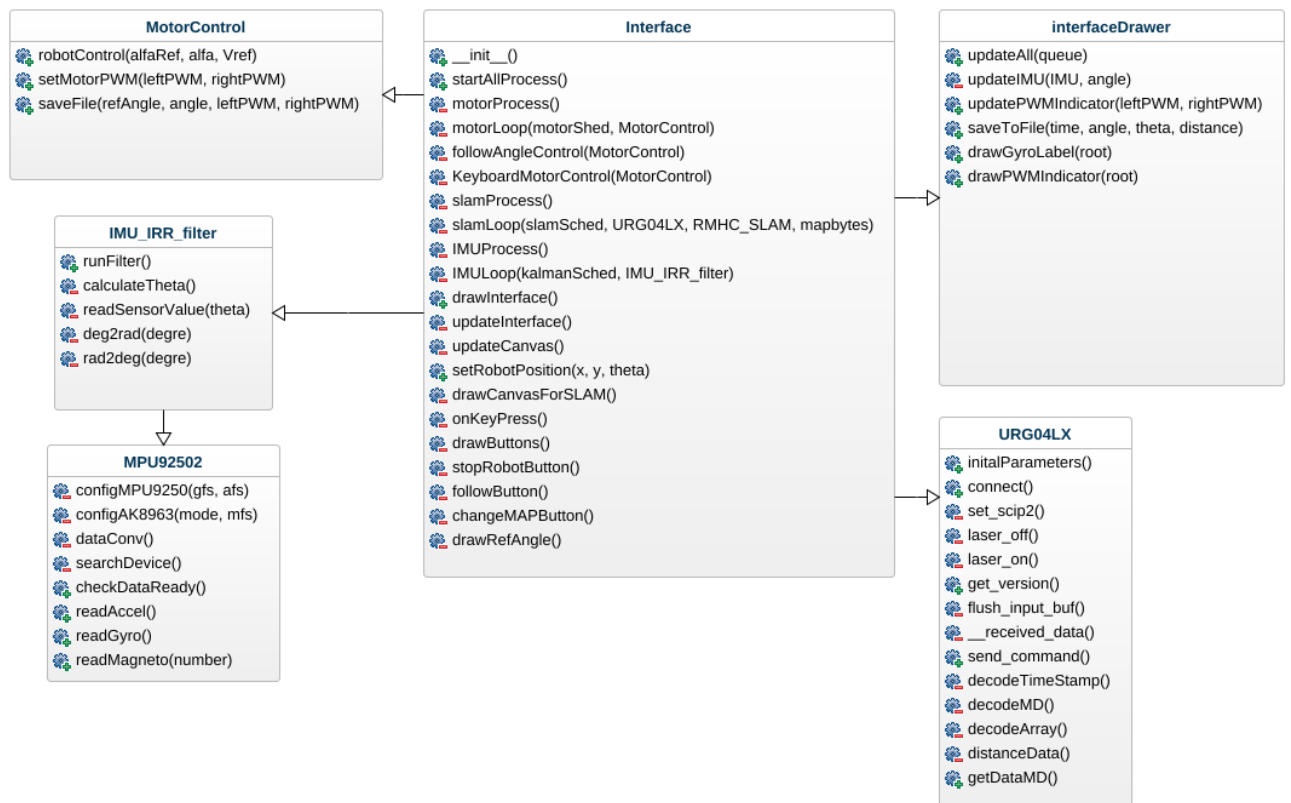
A folyamatok pontos ütemezése érdekében egy sched Python könyvtárat [38] implementáltam, így kellő pontossággal ütemeződnek a folyamatok. Az IMU és a motor folyamata 50 ms -ként, a SLAM és a kezelőfelület 100 ms -ként ütemeződik. A 27. ábrán a mobilis robot aktivitás diagramja látható.



27. ábra: Mobilis robot aktivitás diagramja

Összesen 6 darab üzenetsor biztosítja a folyamatok közti kommunikációt, ezeket az ábrán más-más színnel jelöltem, így könnyebben áttekinthető. Ahogyan az előző ábrán is, pirossal vannak jelölve bekövetkező események és fekete körrel a program kezdete.

A kód áttekinthetősége és könnyű hibaelhárítás érdekében osztályokba zártam a műveleteket. Ezeket az osztályokat a következő ábrán (28. ábra) megtekinthetőek

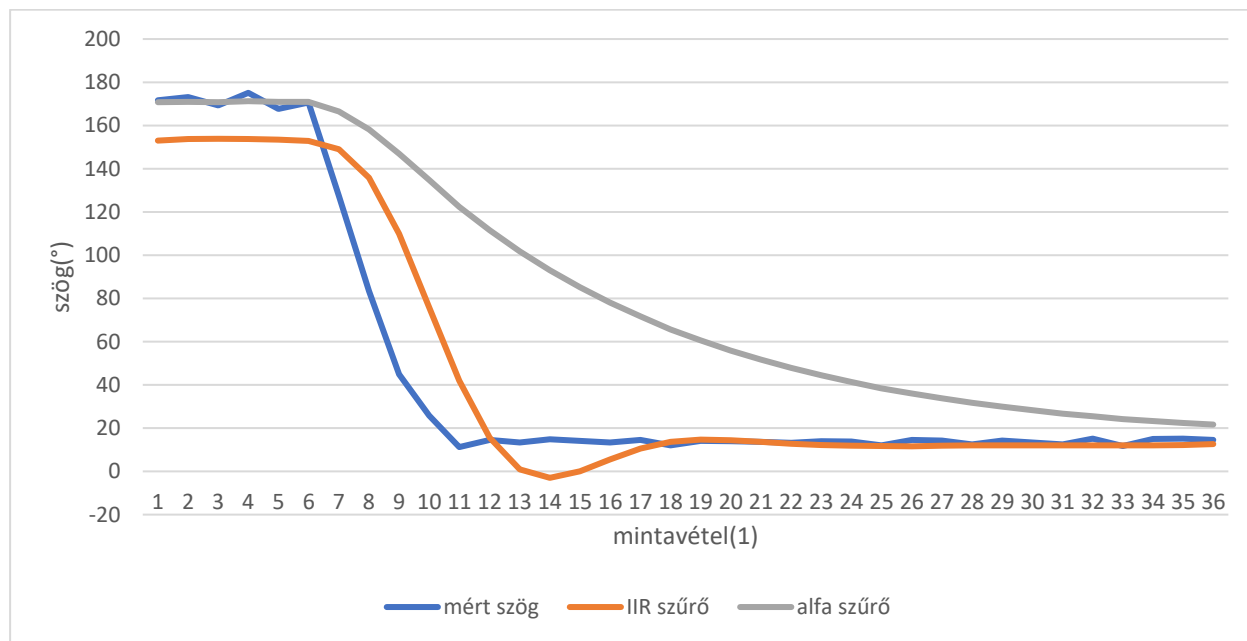


28. ábra: Mobilis robot osztálydiagramja

4. Üzembe helyezés és kísérleti eredmények

4.1. IMU mérések

Készítettem egy egyszerű programot, amely a matplotlib könyvtárat [39] felhasználva ábrázolja a nyers kalibrált szögelfordulást és két szűrt elfordulást. A 29. ábrán piros színnel a nyers adat, kék színnel az IIR szűrővel szűrt és lila színnel az alfa szűrővel szűrt érték látható:

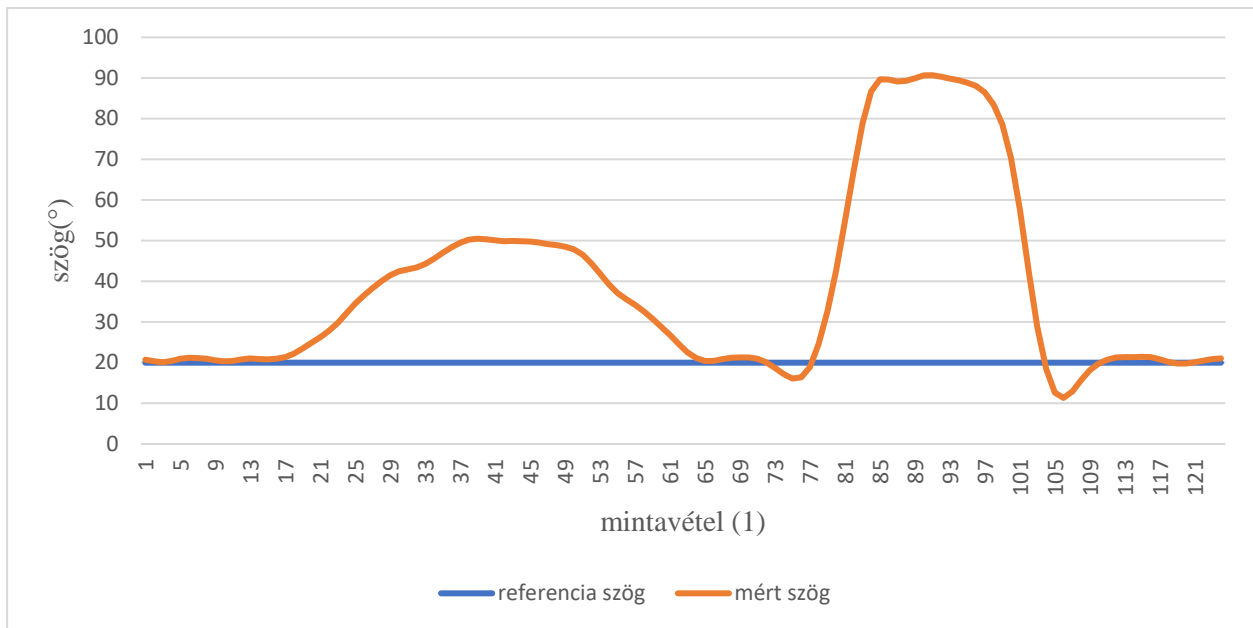


29. ábra: Szűrők összehasonlítása

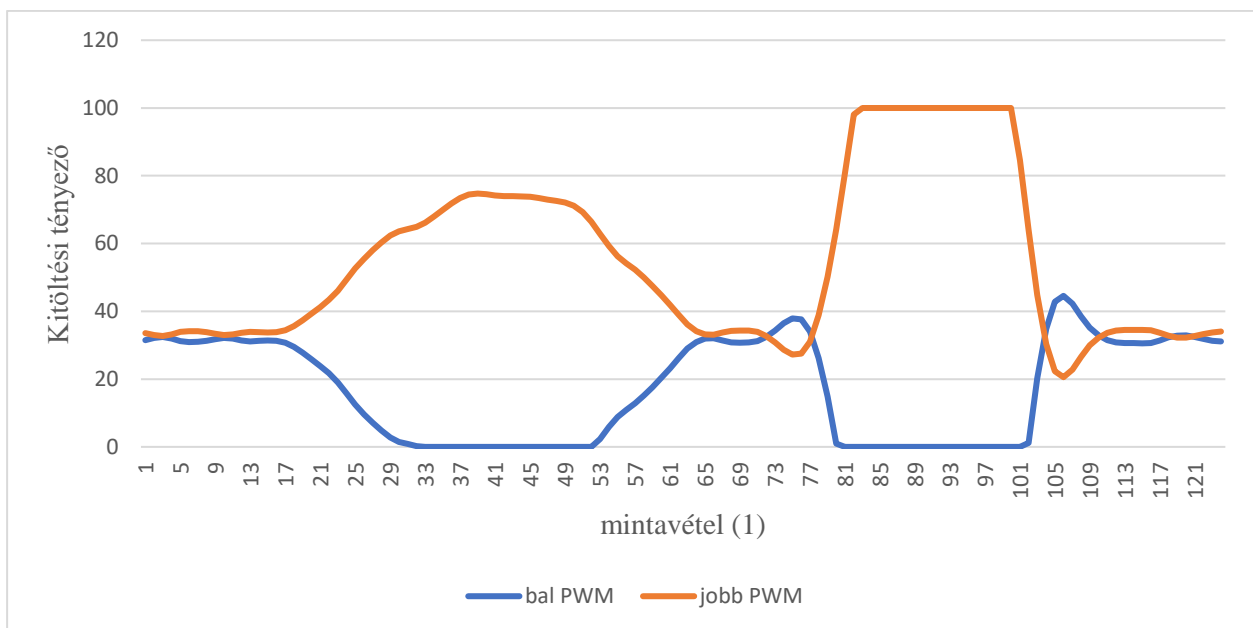
A mérésekből levonható az a következtetés, hogy az IIR szűrők sokkal gyorsabb az átmenet tartománya, viszont egy picivel zajosabb jelet eredményez, mint az Alfa szűrő.

4.2. Motor szabályozás mérése

A mobilis robot szögmenetén való vezérlést megvalósító algoritmus tesztelésekor a függvény bemenetének megadtam konstans referencia szöget, a szűrt orientációt, és elvárt szögsebességet. Az első mérés során a robotot elemelve a talajtól, Z tengely irányába elmozdítottam. A mérési eredményekből is látható, hogy a szabályozó a mobilis robotot próbálja a referencia szög felé vezérelni a mért szög függvényében. Minél nagyobb az eltérés a referencia, valamint a mért szög között, annál nagyobb beavatkozó jelet küld ki a motorvezérlő számára. A 30. és 31. ábrán egy mérés látható, ahol a referencia szög az 20° -ra volt beállítva.

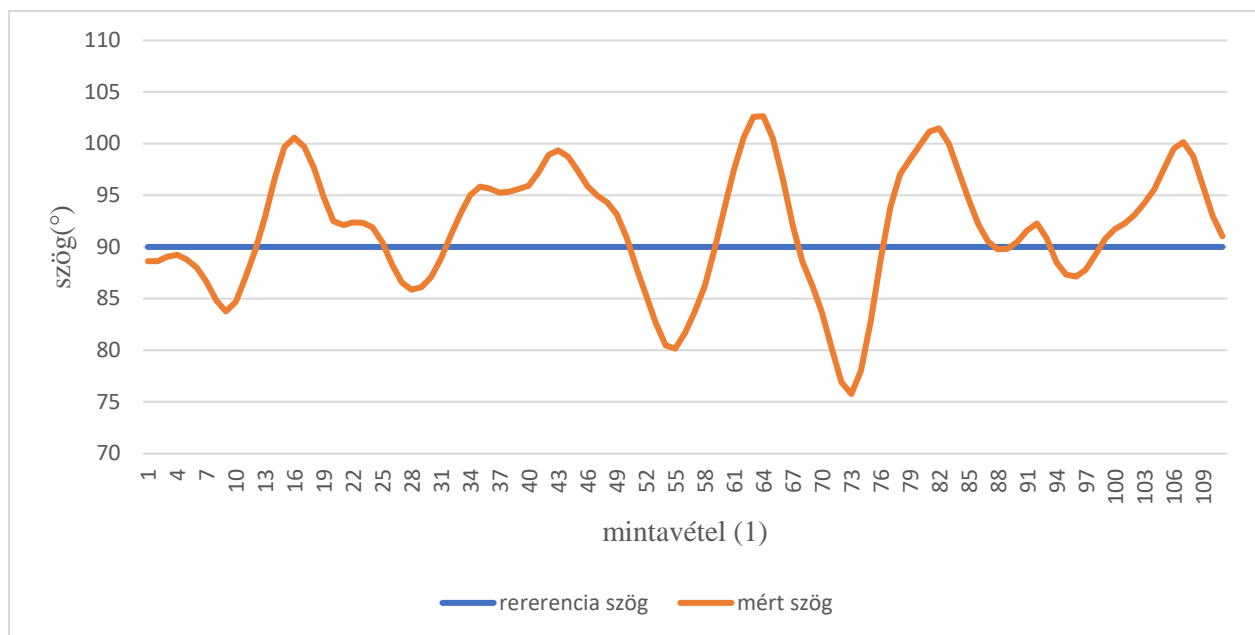


30. ábra: Referencia és mért szög



31. ábra: Beavatkozó jel

A következő mérésnél a robot 3-4m távolságot tudd meg a talajon. Látható, hogy a mért szög enyhén ingadozik a referencia szög körül. Ez a hiba a mérési eredményekből, valamint a villanymotor tulajdonságából eredhet. A 32. és 33. számú ábrán a referenciaszög és mért szög látható, a számú ábrán pedig a beavatkozó jel.



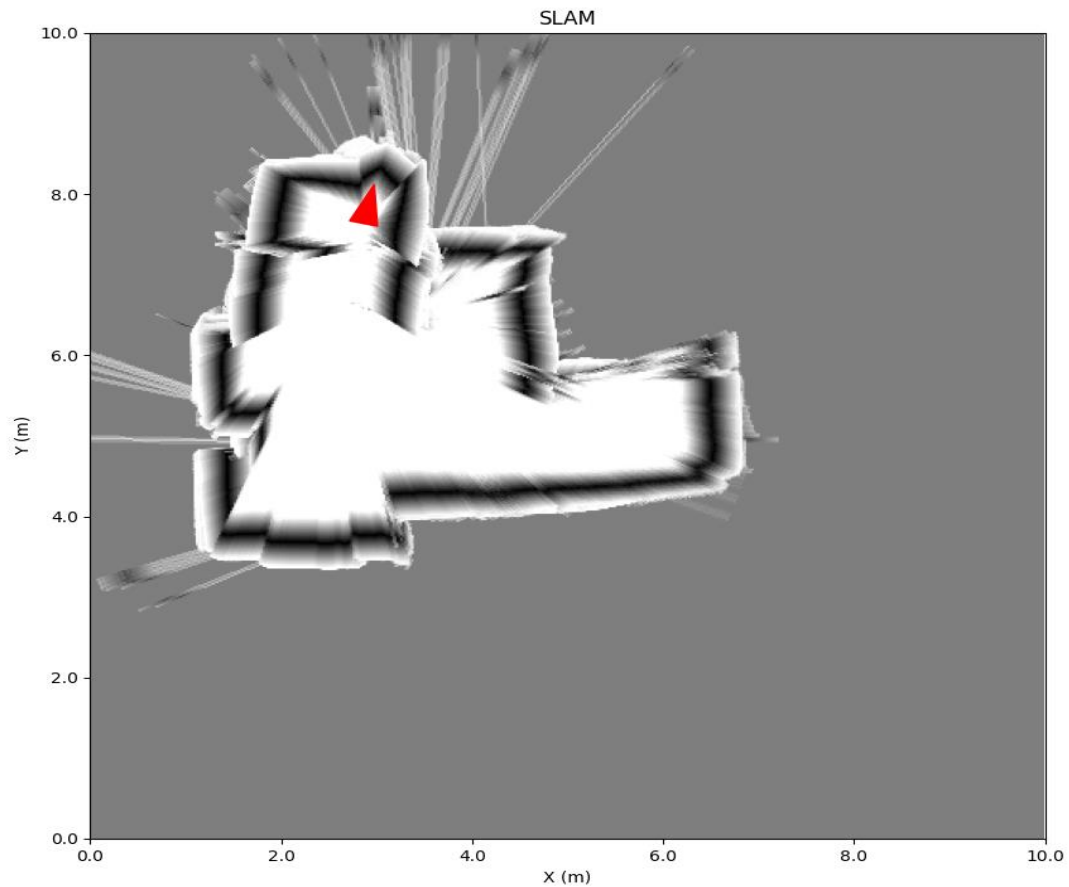
32. ábra: Referencia és mért szög



33. ábra: Beavatkozó jel

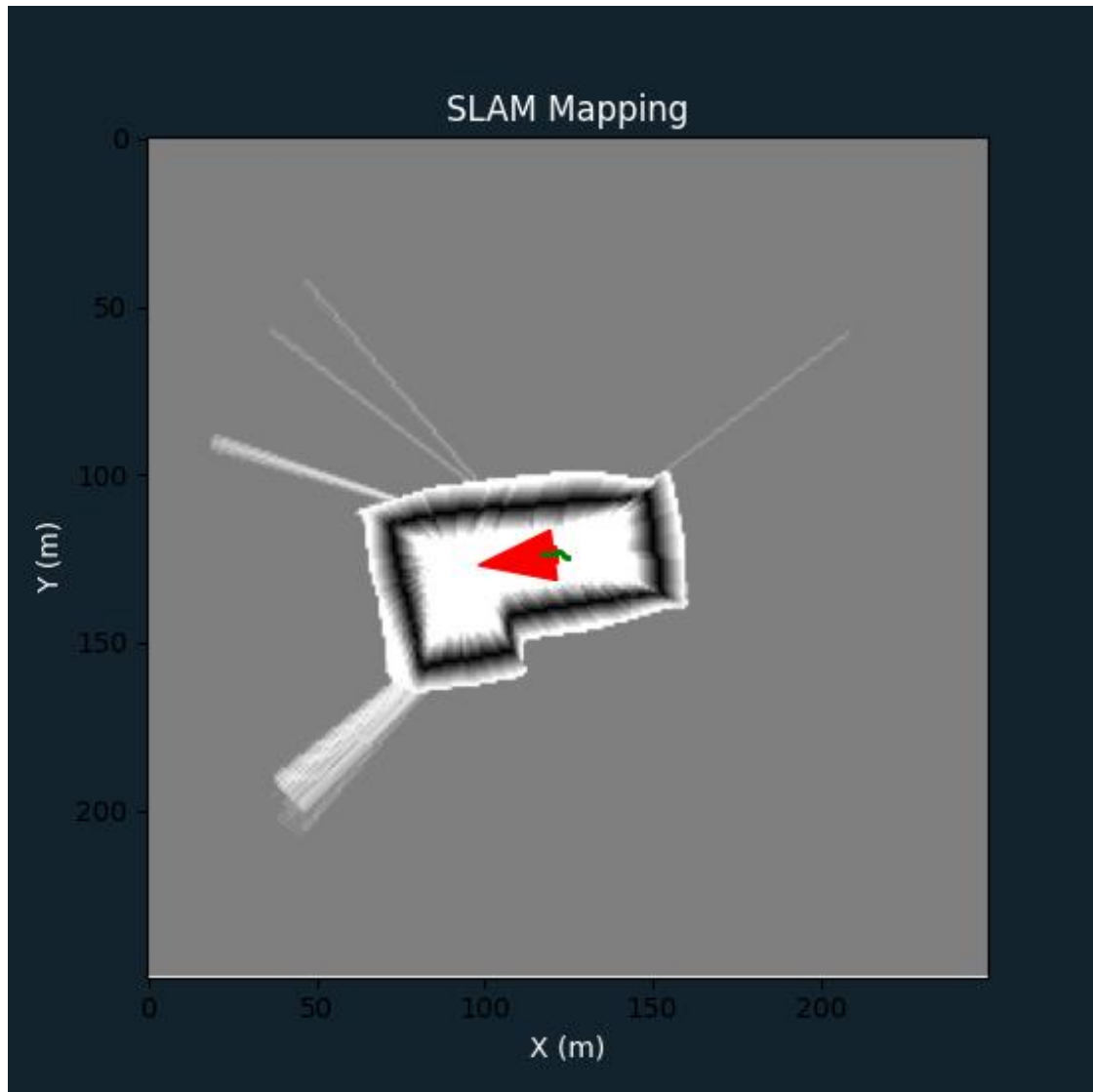
4.3. SLAM mérések

A SLAM mérések során különböző terepadottságú és nagyságú helységet próbáltam feltérképezni. A térképek készítésekor az algoritmus csak a LIDAR érzékelő adatait felhasználva becsülte meg a robot pozícióját és orientációját. A térképek zajosságát nagymértékben javítani lehetne odometria mérések által. A 34. ábrán egy hálószoza térképe látható.



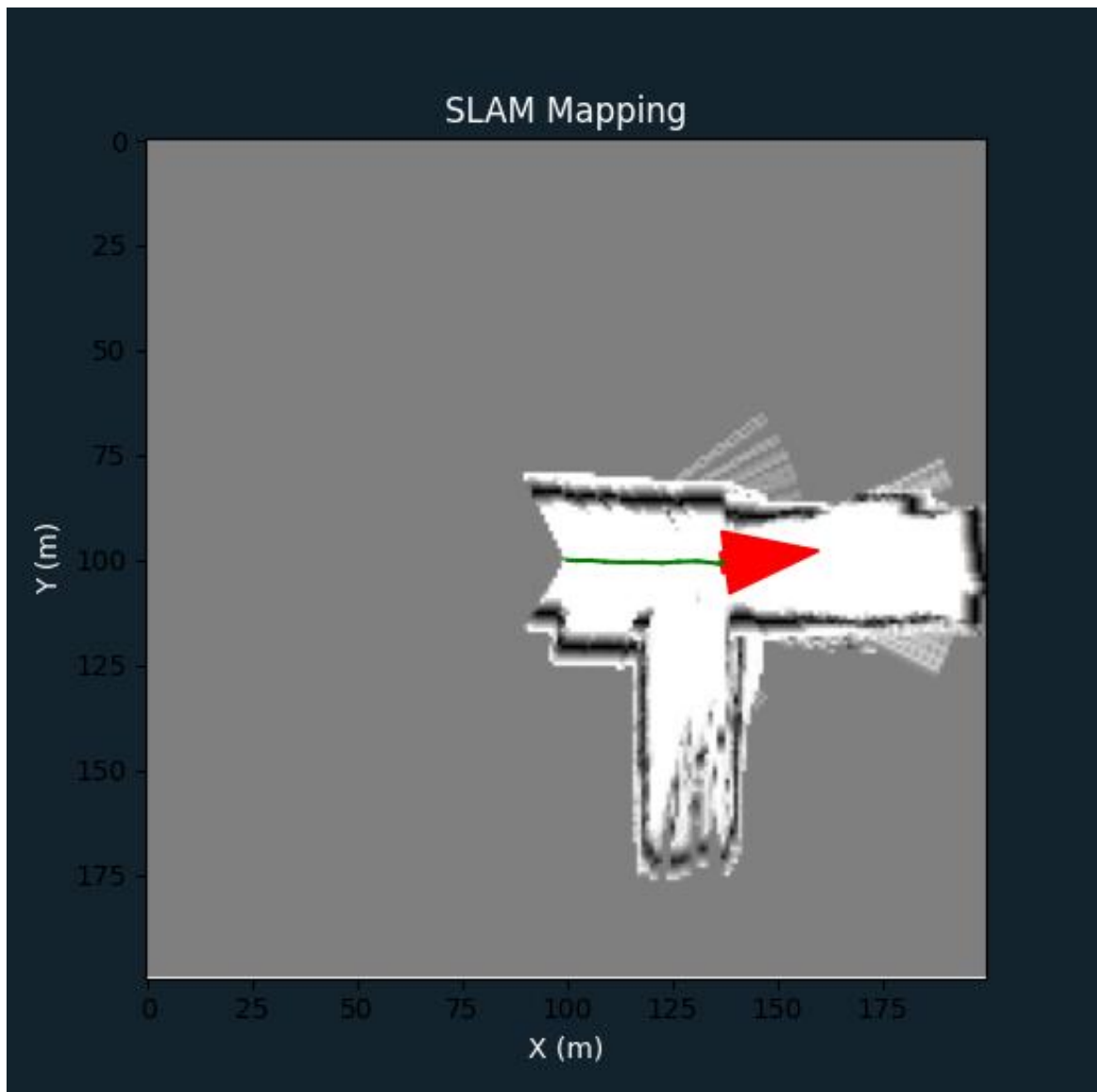
34. ábra: Térkép készítés, mérések 1

A 35. ábrán egy viszonylag kis helyről készítettem térképet. Mivel a LIDAR érzékelő csak 240° - ban képes távolságot mérni, ezért egy szükséges volt a robot elforgatása. Zöld vonallal robot által megtett távolságot jelöltem, jól látható, hogy egy apró fordulással az egész helységet sikerült bemérnem.



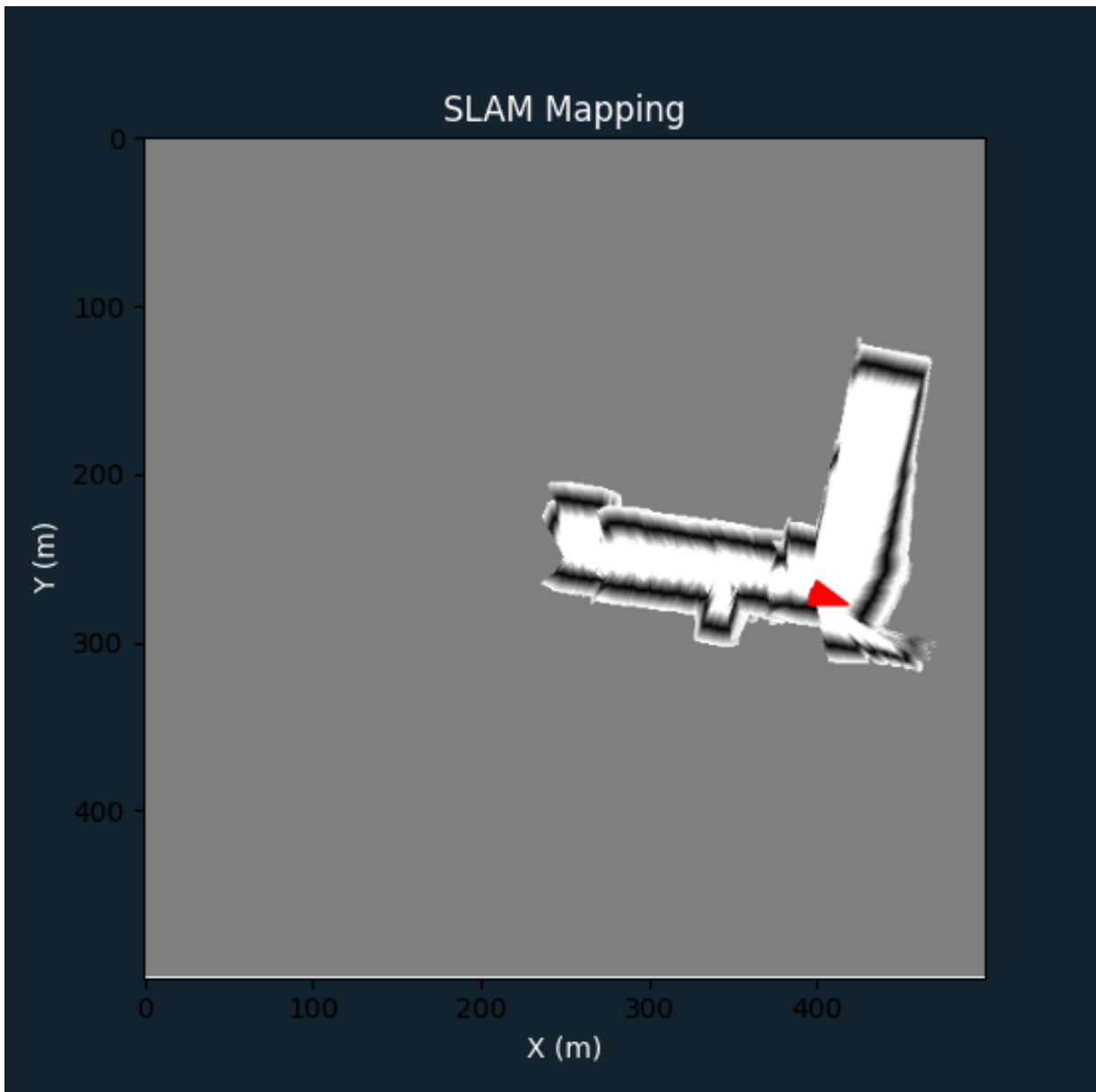
35. ábra: Térkép készítés, mérések 2

A következő mérésnél (36. ábra) felhasználtam a referencia szög mentés vezérlő algoritmust. Látható, hogy a robot egy egyenes mentét haladva készítette el a térképet. A kép alján látható elmosódások annak köszönhetőek, hogy a folyosó mérete túlságosan hosszú volt. A LIDAR érzékelő 4-5 méteres mérési tartományába nem fért már bele.



36. ábra: Térkép készítés, mérések 3

A 37. ábrán szintén egy hosszú folyósót mértem fel.

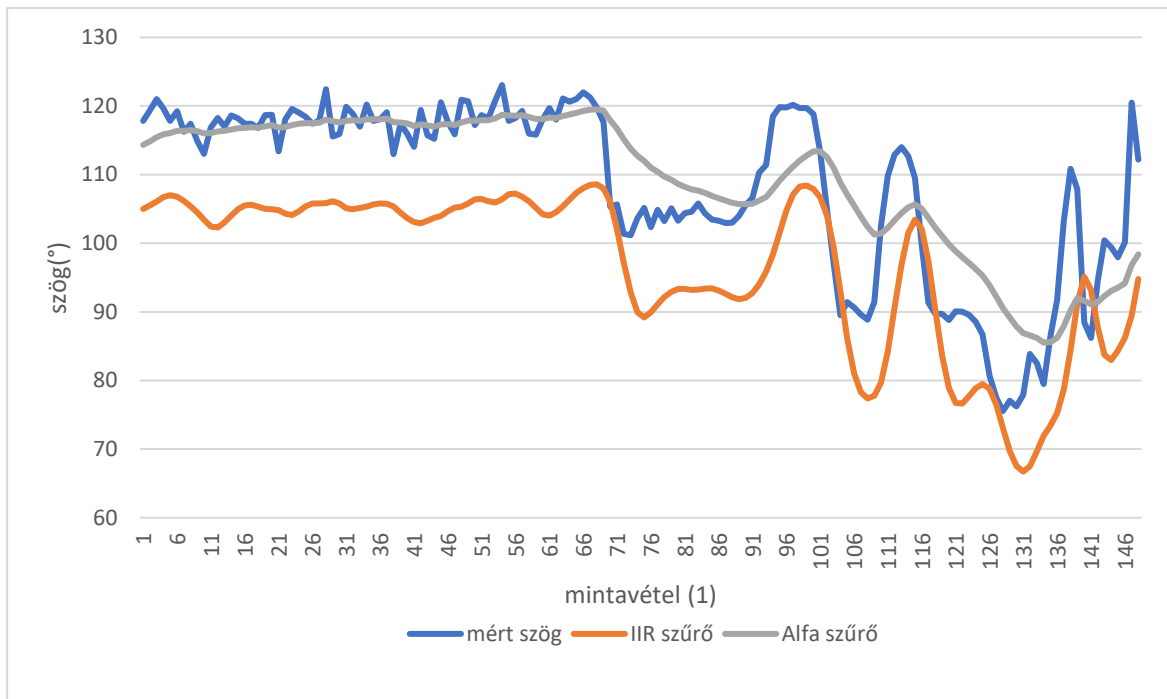


37. ábra: Térkép készítés, mérések 4

4.4. Felmerült problémák és megoldásaik

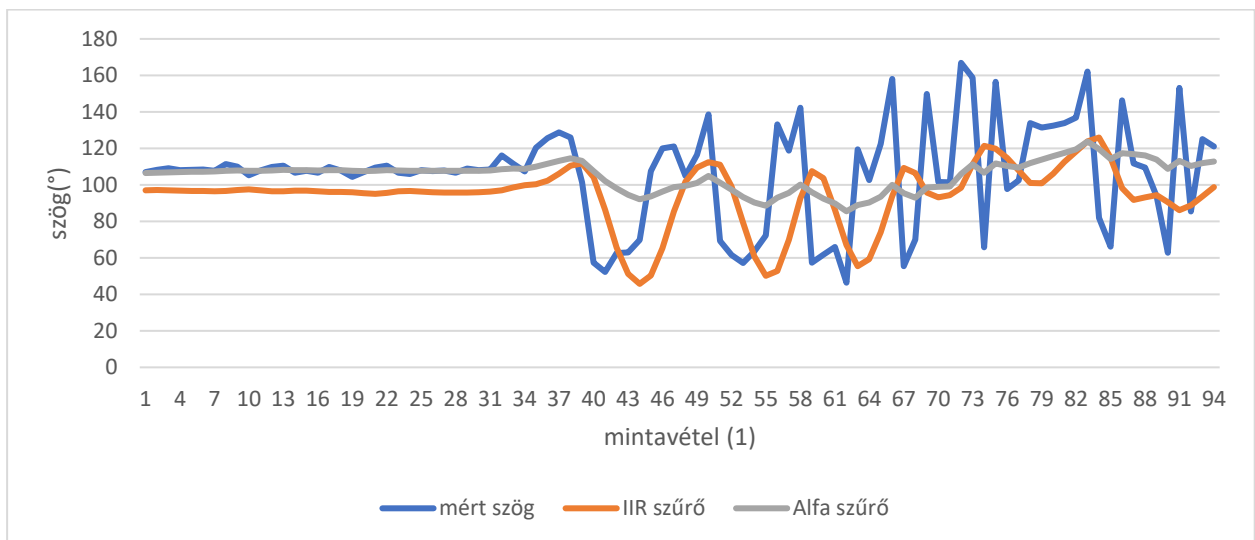
4.4.1. IMU problémák

A IMU mérések a kalibráció ellenére is zajosak bizonyulnak, ha a mobilis robot közelében fém található. Ezeket a problémákat 4.2.2 alpontban tárgyaltam. A következő mérés egy kisebb fém közelében történt. Az 38. ábrán látható, hogy a 66. mintavételben fém került a nyugalmi állapotban lévő robot közelébe, bezavarva az orientációt.



38. ábra: Zajos IMU mérések1

Az egyetemen végzett mérések során, a laborokban található fémasztalok az alábbi ábrán látható módon zavarták az orientáció meghatározását. Az 39. ábrán látható, hogy 34. mintavételben a mobilis robot megközelítette a fémasztalt.

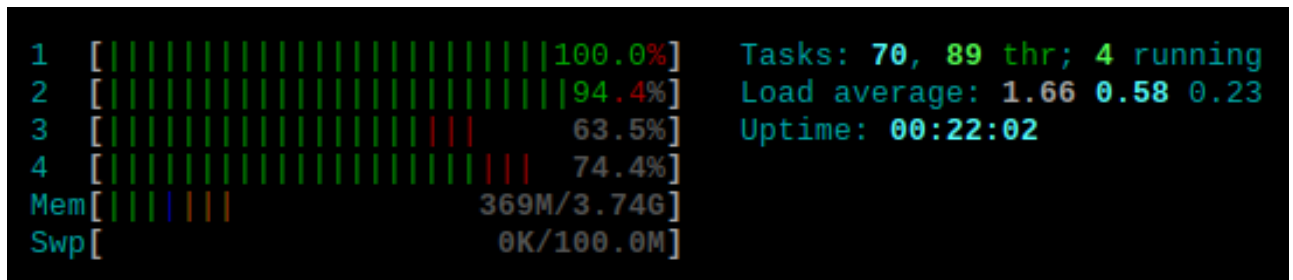


39. ábra: Zajos IMU mérések2

4.4.2. Számításkapacitás

A feladatok folyamatokba helyezése ellenére is a Raspberry Pi 4 számításkapacitás nem bizonyult elégségesnek. A térképkészítéshez 300 000 elemű tömbökkel történő műveletek szükségesek. A térkép előállítása mellett, még a felhasználói felület megjelenítésére is nagy erőforrás szükséges. A problémát úgy oldottam meg, hogy a felhasználói felület frissítési arányát lecsökkentettem, valamint a SLAM algoritmus térkép méretét is. Ezek mellett elhelyeztem egy gombot, amely segítségével kikapcsolhatjuk az online térképrajzolást. Hatékonyabb SLAM algoritmus alkalmazása lenne a leghatékonyabb teljesítménynövelés.

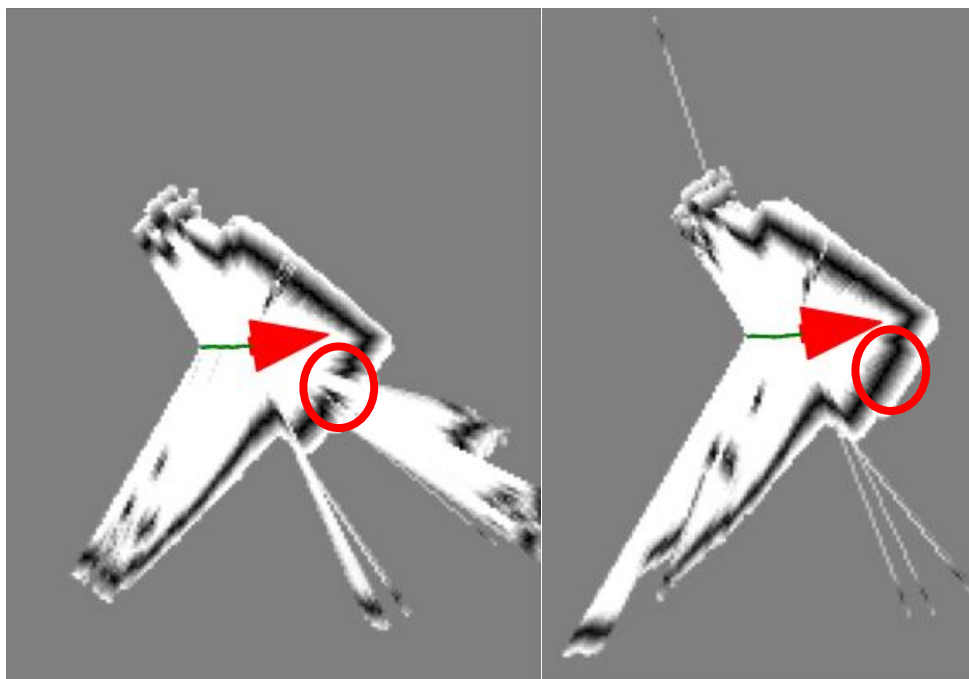
A 40. ábrán a “htop” nevezetű feladatkezelő program látható.



40. ábra: Folyamatfigyelő alkalmazás

4.4.3. LIDAR érzékelő hibái

A LIDAR érzékelő alkalmazása számos előnnyel jár, de a tükröződő felületek könnyen eltorzíthatják a méréseket. A 41. ábrán olyan mérést végeztem el, ahol a robot környezetébe fényes felülettel rendelkező akadályok voltak. A fényes felületet letakarva teljesen más térképkészítési eredményt kaphatunk. Az ábrán piros körrel jelöltem, a fényes felülettel rendelkező akadályok pozícióját a térképen. Ezeket a hibákat más érzékelők, de leginkább kamerák használatával lehetne kiküszöbölni.



41. ábra: Zajos SLAM mérések

5. Következtetések

5.1. Megvalósítások

A mobilis robot megvalósítását a LIDAR érzékelő kommunikációs protokoll megismerésével kezdtem. Az érzékelőtől származó adatok egy saját fejlesztésű szoftverben dinamikusan ábrázoltam. Megvalósítottam motorvezérlő szabályzót, ami segítségével a robot kívánt referenciaszög mentén tudtam vezérelni. Az orientáció meghatározásához elsősorban az I²C kommunikációs protokollt készítettem el, aztán IMU értékeit kalibráltam és szűrtem. A térképkészítéshez egy Breezy SLAM könyvtárat használtam fel. Az alrészek megvalósítása után a részprogramokat összesítettem egy Python nyelven készült felhasználói felület alatt. A párhuzamos feladatvégzés érdekében a feladatok különböző szálakon hajódnak végre. A folyamatok közti kommunikációt üzensorokkal oldottam meg.

5.2. Hasonló rendszerekkel való összehasonlítás

5.2.1. Xiaomi Mi Robot okosporszívóval való összehasonlítás

A porszívó, valamint a mobilis robot között hasonlóság található, úgy hardware, mint szoftver szinten. A két rendszer feldolgozóegysége hasonló architektúrájú processzorból áll. A Xiaomi ARM Coretex A7, a Raspberry Pi 4B ARM Coretex A72 processzorral szerelt. A különbség, hogy a porszívót még másik két adatfeldolgozó egységgel látták el. Ugyan a porszívó több érzékelővel van ellátva, de a környezet letapogatásra szintén LIDAR érzékelőt használ. Mind a két rendszer egyfajta SLAM algoritmuson alapszik. Az okosporszívó képes autonóm navigációra a feltérképezett helységben, valamint saját telefonos applikációval is rendelkezik, ahol a felhasználó felügyelheti, időzítheti a működési ciklusait. Ezek mellett a porszívó az energiaellátásra is odafigyel, amint merülni kezd dokkoló állomásához navigál, hogy feltöltse akkumulátorait.

5.3. Továbbfejlesztési lehetőségek

A jövőbeli terveim közé tartozik, a LIDAR érzékelőből származó pontfelhőt egy hatékonyabb SLAM algoritmus segítségével dolgozzam fel, hogy jobb térképkészítési eredményeket kapjak. Odometria alkalmazása a mobilis roboton, így sebessége és elfordulása meghatározható lenne. A felhasználó számára, egy telefonos applikáció készítése, valamint a C# - ban készített felhasználói felület tovább gondolása egy kiváló továbbfejlesztési lehetőség. A mobilis robot végleges szerkezeti felépítésének 3D tervezése és nyomtatása.

6. Köszönetnyilvánítás

Szeretnék köszönetet mondani témavezetőimnek, Dr. Márton Lőrinc egyetemi tanárnak és Fehér Áron tanársegédnek, hogy gyakorlati és elméleti információkkal segítettek a munkámat, valamint, hogy tapasztalatukat és észrevételüket megosztva útmutatását nyújtottak a dolgozatom elkészítéséhez.

7. Irodalomjegyzék

- [1] Map Comparison of Lidar-based 2D SLAM Algorithms Using Precise Ground Truth: IEEE, 2018
- [2] He, Ray C. Stereo Vision and Mapping with Unsynchronized Cameras. Massachusetts : MIT, 2008
- [3] G.N Desouza, A.C. Kak: Vision for mobile robot navigation: a survey, IEEE Transaction on Pattern Analysis and Machine Intelligence, Vol. 24, Feb 2002
- [4] „Robot navigation”, Wikipedia, Wikimedia Foundation, 30 May. 2021, 1025885515, https://en.wikipedia.org/wiki/Robot_navigation.
- [5] Dr. Márton Lőrinc, Kereken gördülő mobilis robotok irányítása: https://ms.sapientia.ro/~martonl/Docs/Lectures/Mobilis_Robotok_Iranyitasa.pdf
- [6] „Single-board computer”, Wikipedia, Wikimedia Foundation, 20 April 2021, 1018888075, https://en.wikipedia.org/wiki/Single-board_computer.
- [7] „Raspberry Pi”, Wikipedia, Wikimedia Foundation, 26 June 2021, 1030470581, https://en.wikipedia.org/wiki/Raspberry_Pi.
- [8] „H – bridge”, Wikipedia, Wikimedia Foundation, 24 June 2021, 1030262656, <https://en.wikipedia.org/wiki/H-bridge>.
- [9] STMicroelectronics, L292N Datasheet, <https://www.alldatasheet.com/datasheet-pdf/pdf/22429/STMICROELECTRONICS/L292.html>
- [10] „Sensor”, Wikipedia, Wikimedia Foundation, 10 April 2021, 1017101243, <https://en.wikipedia.org/wiki/Sensor/>
- [11] „Laser”, Wikipedia, Wikimedia Foundation 21 June 2021, 1029664500, <https://en.wikipedia.org/wiki/Laser>
- [12] „LIDAR”, Wikipedia, Wikimedia Foundation, 21 June 2021, 1029665921, <https://en.wikipedia.org/wiki/Lidar>
- [13] HOKUYO URG 04LX-UG01 Datasheet, https://www.hokuyo-aut.jp/member/download_data.php?category=4
- [14] „Inercial measurment unit”, Wikipedia, Wikimedia Foundation, 28 May 2021, 1025660823, https://en.wikipedia.org/wiki/Inertial_measurement_unit
- [15] MPU9250 Datasheet, <https://datasheet.octopart.com/MPU-9250-InvenSense-datasheet-21859386.pdf>
- [16] “Kommunikációs protokollok: jellemzők, típusok, példák”, <https://hu.warbletoncouncil.org/protocolos-de-comunicacion-4917>

- [17] „Basics of the I2C communication protocol”,
<https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/>
- [18] Wolfram Burgard, Cyrill Stachniss, Kai Arras, Maren Bennewitz: Introduction to Mobile Robotics, SLAM: Simultaneous Localization and Mapping
- [19] „Simultaneous localization and mapping”, Wikipedia, Wikimedia Foundation, 8 June 2021, 1027469336,
https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping.
- [20] „Párhuzamos rendszerek. Párhuzamos adatfeldolgozás az EUM-en”,
Kotovsk2010, <https://viws.ru/hu/parallelnye-sistemy-parallelnaya-obrabotka-dannyh-na.html>
- [21] Philippa Young, „Designing an Analogue Low Pass Filter System for Quantum Ion Trap”, Summer Research Placement 2008, <http://users.sussex.ac.uk/~pjly20/ras100.html>
- [22] Natalie Red, „Viomi V2 vs Roborock S6: The Differences”, 11 Dec. 2020,
<https://smartrobotreviews.com/reviews/roborock-s6-vs-viomi-v2.html>
- [23] Volkswagen AG, „Az önvezetés öt szintje”, <https://www.volkswagen.hu/e-mobilitas-es-id-hub/id-hub/id-magazin/az-onvezetes-5-szintje>
- [24] William Gallagher, „How to use the LIDAR scanner in iPhone 12 Pro”, 05 Mar. 2021, <https://appleinsider.com/articles/21/03/02/how-to-use-the-lidar-scanner-in-iphone-12-pro>
- [25] Shimakage, Akira-Sasaki, „FaBo9AXIS-MPU9250-Python”, 2019,
<https://github.com/FaBoPlatform/FaBo9AXIS-MPU9250-Python>
- [26] Karl-Petter Lindegaard, „smbus2 0.4.1”, MIT License (MIT), 17 Jan. 2021,
<https://pypi.org/project/smbus2/>
- [27] Riki webpage, „Calibrating the magnetometer”, 28 July, 2018,
<https://thepoorengineer.com/en/calibrating-the-magnetometer/#Calibration>
- [28] Diagramok készítése: <https://www.genmymodel.com/>
- [29] Python Software Foundation, „signal – set handler for asynchronous events”, 27 Jun. 2021, <https://docs.python.org/3/library/signal.html>
- [30] Simon D. Levy, „Breezy SLAM”, 13. Jan. 2019,
<https://github.com/simondlevy/BreezySLAM>
- [31] Bresenham’s Line Generation
https://www.tutorialspoint.com/computer_graphics/line_generation_algorithm.htm

- [32] „Particle filter”, Wikipedia, Wikimedia Foundation, 10 June 2021, 1027788024
https://en.wikipedia.org/wiki/Particle_filter
- [33] „Bayesian inference” Wikipedia, Wikimedia Foundation, 28 June 2021,
1028448848, https://en.wikipedia.org/wiki/Bayesian_inference
- [34] Matlab „stateEstimatorPF”
<https://www.mathworks.com/help/nav/ref/stateestimatorpf.html#bu31dpn-6>
- [35] „Monte Carlo localization”, Wikipedia, Wikimedia Foundation, 28 June 2021,
997855519, https://en.wikipedia.org/wiki/Monte_Carlo_localization#Sensor_update
- [36] „Multithreading vs Multiprocessing: What’s the difference?”,
<https://www.guru99.com/difference-between-multiprocessing-and-multithreading.html>
- [37] Python Software Foundation, „queue”, 27 Jun 2021,
<https://docs.python.org/3/library/queue.html>
- [38] Python Software Foundation, „sched – Event scheduler”, 27 Jun 2021,
<https://docs.python.org/3/library/sched.html>
- [39] John Hunter, Darren Dale, Eric Firing, Michael Droettboom, „Matplotlib:
Visualization with Python”, 08 May. 2021, <https://matplotlib.org/>
- [40] Peter Corke, Robotics, Vision and Control Fundamental Algorithms in
MATLAB, Springer, 2011

UNIVERSITATEA SAPIENTIA DIN CLUJ-NAPOCA
FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE, TÎRGU-MUREȘ
SPECIALIZAREA AUTOMATICĂ ȘI INFORMATICĂ APLICATĂ

Vizat decan
Conf. dr. ing. Domokos József

Vizat director departament
Ș.l. dr. ing Szabó László Zsolt