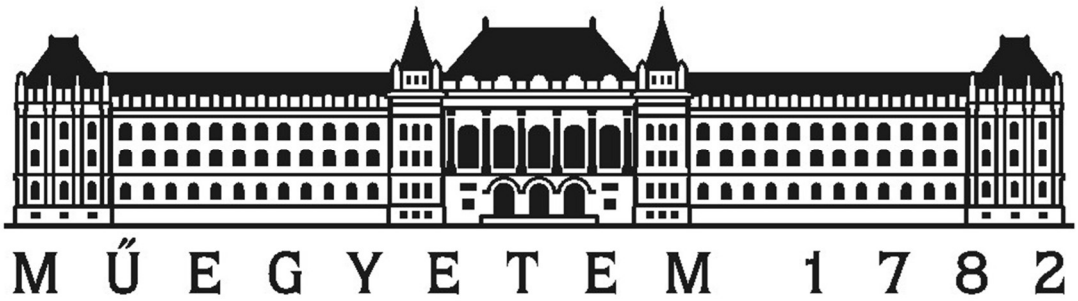


Budapesti Műszaki és Gazdaságtudományi Egyetem

Építőmérnöki kar

Mobil térképező platform építése és programozása

Konzulensek:

Dr. Barsi Árpád

Dr. Siki Zoltán

Készítette:

Horváth Viktor Győző

Tartalom

Tartalom 2
Absztrakt 3
1.Mobil térképezés és története 3
2.ROS – Robot Operating System 4
Az ROS fogalmi szintjei: 5
3.SLAM algoritmusok....7
Az ROS-ben elérhető SLAM algoritmusok: 8
HectorSLAM algoritmus matematikája 9
4.Platform hardveres megvalósítása 11
5.Szoftveres megvalósítás 22
a.Lézerszkennelés és SLAM 24
b. Irányítás megvalósítása pythonban....27
c. Kamera felhasználása az irányításhoz....28
d. Mérési eredmények kinyerése 28
6.Mérési eredmények, alkalmazás 29
7.Konklúzió, fejlesztési lehetőségek, gyengeségek 39
a.Konklúzió 39
b.Fejlesztési irányok 39
c.A rendszer hátulütői 39
8.Köszönetnyilvánítás....39
9.Irodalomjegyzék 41
Mellékletek....42

1. Absztrakt

Dolgozatomban egy nyílt hardver komponensekből épített saját mobil térképező rendszer megépítését és programozását mutatom be, majd az eszközzel készített térképek pontosságát és megbízhatóságát vizsgálom. A platformom egy RPlidar 360° lézerszkennerből, egy Raspberry Pi 4-ből, két motorból, és egy motorvezérlő lapkából, 2 3500mAh akkumulátorból áll. A platform WiFin keresztül vezérelhető, ehhez egy VNC szervert telepítettem a Raspberryre. A platformon Ubuntu Mate 20.04 (Focal Fossa) operációs rendszer fut, a robot vezérléséért pedig az Robot Operating System (ROS) úgynevezett „middleware" felelős, ennek is a legújabb stabil verziója a Noetic Ninjemys. A térképezéshez Hector SLAM algoritmust használok, ami a Heterogeneous Cooperating Team Of Robots (Heterogén robotok együttműködése) rövidítése. Az eljárást a Darmstadti Műszaki Egyetemen (TU Darmstadt) városi keresés és mentés (Urban Search and Rescue) feladatokra fejlesztették ki. Az algoritmus nagy előnye, hogy nincs szükség odometria adatokra, és inerciális mérőegységre (IMU) sem a platformon. Nyilvánvalóan az odometria hiányának is megvannak a hátulütői, dolgozatomban erre is mutatok példát.

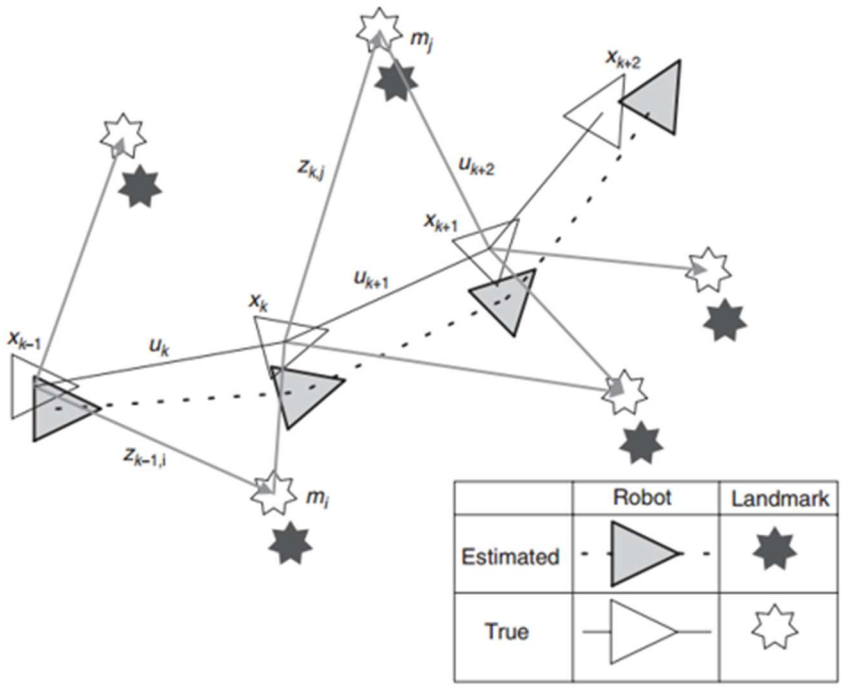
2. Mobil térképezés és története

A Simultaneous Localization And Mapping (egyidejű helymeghatározás és térképezés, SLAM) problémája egyidős a robotika megjelenésével. A robotnak nem csupán egy helyet kell felmérni és abból egy térképet produkálni, hanem ezt anélkül kell tennie, hogy információval rendelkezne arról, hogy a platform maga hol tartózkodik. Ezt még nagyságrendekkel bonyolíthatja, hogy a környezetet statikusnak tekintjük, vagy az változik a térképezés ideje alatt. Statisztikai oldalról nézve a térképezés egy Bayes következtetési probléma, csakúgy, mint a lokalizáció. Ennek a problémának a leggyakoribb megoldása a Bővített Kálmán-szűrő (EKF), de léteznek más alapon működő algoritmusok is. Ezekre dolgozatomban később bővebben is kitérek.

Kálmán-szűrőt használtak a NASA Apollo programjának navigációs számítógépében is, amivel a rakéta nemlineáris röppályáját számították.

Az első valódi általános célú SLAM-mel és mesterséges intelligenciával működő robotnak a Shakey-t tekintik. Shakey-t, a robotot 1966 és 1972 közt fejlesztették a Stanfordi Egyetem Mesterséges Intelligencia Kutatóközpontjában. A projekt felhasználta a robotika, a gépi látás (CV) kutatási eredményeit. Ennek a projektnek köszönhető például az A* útkereső algoritmus, a Hough-transzformáció és a láthatósági gráf is.

Dolgozatomban vizsgálom, hogy a SLAM-mel előállított, robot navigációjára szolgáló térképeket mennyire lehet geodéziai célokra felhasználni.



1. ábra A SLAM probléma szemléltetése, a robot helyzetének és a tereptárgyak helyzetének szimultán becslése (https://blog.acolyer.org/)

3. ROS – Robot Operating System

Az ROS a Robot Operating System rövidítése. Az első verzióját 2007-ben adták ki, azóta folyamatosan fejlesztik. A legújabb kiadása 2019 november 22-én jelent meg. Az ROS megnevezése operációs rendszer, de valójában meta-operációs rendszer; ez alatt azt értjük, hogy olyan feladatokat is ellát, amik klasszikusan az operációs rendszer feladatai, de nem helyettesíti az operációs rendszert. Ilyen feladatok például a hardver absztrakció, az alacsony szintű eszközvezérlés és a gyakori funkcionalitások implementációja. A felhasználók számára eszközöket és könyvtárakat biztosít még, amelyek segítenek a kódok beszerzésében, építésében, írásában és futtatásában akár több számítógépen is.

Hasonló robot keretrendszerek:

- Player,
- YARP,
- Orca,
- Microsoft Robotics Studio.

Az ROS leírható, mint peer-to-peer számítógépes folyamatok hálózata, amelyeket összekapcsol az ROS kommunikációs infrastruktúrája. Az ROS lehetővé tesz többfajta kommunikációt is, ide tartoznak az RPC-szerű (Remote Process Calling – távoli eljárás hívás) kommunikációs formák, az aszinkron adatfolyamok és az adatok tárolása, egy úgynevezett „paraméter szerveren”. Az ROS ugyan nem valós idejű keretrendszer, de

lehetséges valós idejű kódok integrálása az ROS-ba. Ezeket a funkciókat a 2.0-ás kiadással nagyban javították, 2016 végén.

Az ROS keretrendszerben írt szoftverek 3 fő csoportba sorolhatók:

- nyelv- és platformfüggetlen eszközök ROS alapú szoftverek építéséhez és terjesztéséhez,
- ROS kliens könyvtárak,
- „csomagok”, amikben applikációhoz kötődő kódok vannak, melyek egy vagy több ROS kliens könyvtárat is használnak.

A fő ROS kliens könyvtárak Unix-alapú rendszerek felé irányulnak, ennek fő oka a nagy fokú függőség a nyílt forráskódú szoftvertől. Ezekhez a könyvtárakhoz az Ubuntu Linux a támogatott operációs rendszer, míg a macOS, Fedora Linux és a Microsoft Windows csak kísérleti jellegűek és közösségi támogatással operálnak. A natív Java könyvtárnak viszont nincsenek ilyenfajta megkötései, így az használható Androidos fejlesztéshez is. Létezik Javascript könyvtár is, ami lehetővé teszi a böngészőben való futtatást is.

Az ROS fogalmi szintjei:

Az ROS-nek 3 fogalmi szintje létezik:

- *fájlrendszer szint*
- *számítási folyamatábra szint*
- *közösségi szint*

Az ROS *fájlrendszer szinten* a következőkből áll:

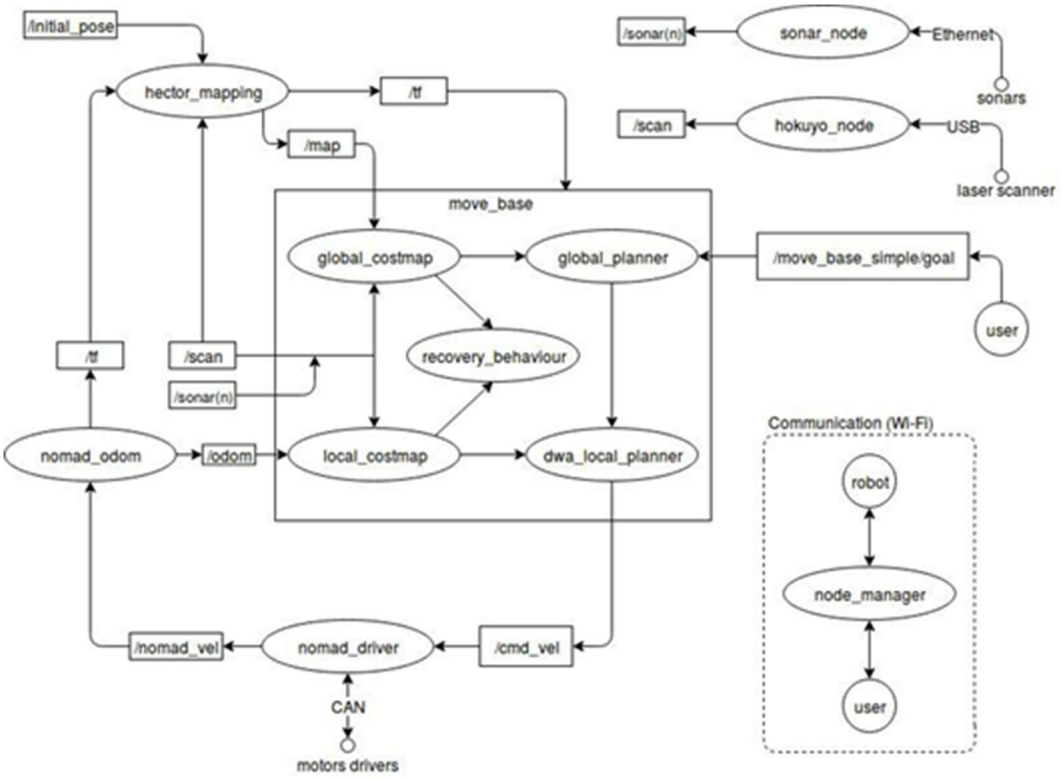
- Packages – csomagok, ezek egy ROS alapú szoftver fő egységei; egy csomag tartalmazhat folyamatokat, könyvtárakat, adatbázisokat és konfigurációs fájlokat vagy bármit, amíg ezek egy használható egységet alkotnak. A csomagok rendezett egésze alkot egy stacket.
- Metapackages – metacsomagok, speciális csomagok,
- Package Manifests – csomagjegyzék, xml formátumban metaadatokat szolgáltatnak egy csomagról,
- Repositories – raktárak, csomaggyűjtemények, melyeknek verziókezelése azonos,
- Message types – üzenettípusok, üzenetek leírói, az ROS-ban küldött üzenetek adatstruktúráit definiálja,
- Service types – szolgáltatástípusok, szolgáltatások leírása, a szolgáltatások adatkérés és fogadás adatstruktúrái.

ROS *számítási folyamatábra szinten* a következőkből áll:

- Nodes – csomópontok, csomópont alatt a folyamatokat értjük, amik számításokat végeznek. Az ROS-t modulárisnak tervezték, egy robotirányító rendszer általában nagyon sok csomópontból tevődik össze.

- Master – irányító, ez végzi a névregisztrációt és a keresést a számítási folyamatban, az irányító nélkül a csomópontok nem lennének képesek kommunikációra, vagy szolgáltatások meghívására.
- Parameter server – paraméter szerver, lehetővé teszi adatok tárolását egy központi helyen az adatbázis kulcsok alapján,
- Messages – üzenetek, a csomópontok üzenetekkel kommunikálnak egymással, egy üzenet csak egy előre meghatározott adatstruktúra,
- Topics – témák, az üzeneteket egy közzététel/feliratkozás modellel juttatják célba. Ha egy csomópont valamilyen bemeneti adatot keres, feliratkozhat az adott témára, és megkapja az azt szolgáltató csomóponttól vagy csomópontoktól. Egy csomópont nyilván lehet egyszerre közzétevő és feliratkozó is, akár több témában is. A témákra azért van szükség, hogy az adatok létrehozását és felhasználását külön válasszák.
- Services – szolgáltatások, az előző publish/subscribe modell nagyon flexibilis, de mivel több-a-többhöz típusú, és egyirányú kommunikáció, így nem alkalmas egyszerű kérés-válasz protokollra, a kérés-válasz kétirányú kommunikációt teszik lehetővé a service-ek
- Bags – táskák, egy olyan formátum, amellyel az üzenetek tartalmát menthetjük és kérhetjük vissza.

Alább látható egy példa a számítási folyamatábrára; az egyes elemek feliratából könnyedén visszafejthető a robot logikája:



2. ábra Számítási folyamatábra (Słomiany et al. 2020)

ROS *közösségi szinten* a következőkből áll:

- Distributions – disztribúció, verziószámmal ellátott stackek telepíthető gyűjteménye,
- Repositories – raktárak, az ROS egymáshoz kapcsolódó kódraktárak hálózatára támaszkodik, ahol különböző szervezetek fejleszthetik a saját robot szoftver komponenseiket,
- ROS Wiki – az ROS közösség fő kommunikációs platformja, és az ROS fő dokumentációja,
- Bug Ticket rendszer – hibajelentésekre szolgáló rendszer,
- ROS Answer – kérdezz-felelek weboldal ROS-hez kapcsolódó kérdésekhez,
- Levelezőlisták.

4. SLAM algoritmusok

A SLAM a robotika egyik legkutatottabb területe. Roppant hasznos térképek létrehozására és frissítésére ismeretlen környezetekben. Napjainkban a SLAM legnagyobb kihívásai, hogy szenzoraink hibái összeadódnak a robot mozgása közben, a térképezett tér gyors változása, a szenzoraink által két különböző időpontban észlelt mérés ugyanarra az objektumra vonatkozik-e és a környezet állandó. (Santos et al. - 2013) A SLAM robotok talán legnagyobb előnye, hogy olyan helyről is kaphatunk információt, ahova embernek kockázatos lenne belépnie.

A feldolgozott szakirodalomban ötféle SLAM technikát értékelnek ki, amelyek elérhetők az ROS keretrendszerben. Ezeket a technikákat kétdimenzióban és előre épített környezetben egy Arduino-alapú roboton értékelték ki.

Korunkban az összes elismert SLAM-algoritmus valószínűségeken alapul. A valószínűségek alkalmazásának nagy előnye a robusztusság a mérési zajra és a mérések bizonytalanságának számszerűsítésének képessége. A térképezésre használt valószínűségi modellek a problémák megoldásában a Bayes-tételre hagyatkoznak.

A Kálmán-szűrők a Bayes-i szűrők legnépszerűbb alkalmazásai. A Kálmán-szűrőknek két fő lépése van: a predikció és a korrekció. A predikció egy korábbi iterációból számítja a pozíciót, míg a korrekciós lépésben a predikcióból származó pozíciót kombinálják a szenzorokból érkező adatokkal. Így beszélhetünk a prior és a posterior pozícióról. A robot helyzetének modelljében fellelhető nemlinearitási problémákat Bővített Kálmán-szűrő alkalmazásával lehet figyelembe venni.

A részecske szűrők is a Bayes-szűrők egy alkalmazási típusai. A posterior valószínűséget részecskék egy súlyozott gyűjteménye mutatja, ahol minden részecskének van egy fontossági rangja. A részecske szűrők azt feltételezik, hogy a következő állapot, csak az előző állapottól függ (Markov-feltételezés). Kezdetben a szűrő azt feltételezi, hogy a pozíció bárhol lehet, majd a Bayes-tétel alkalmazása után a legkisebb valószínűségű pozíciókat eldobja. A részecske szűrők előnye, hogy a bizonytalanságot multimodális eloszlásokon keresztül mutatja, és elbír a nem-Gauss eloszlású zajjal.

Létezik egy FastSLAM nevű eljárás is, ami a két módszert kombinálja; egy módosított részecske szűrőt (particle filter) használ, amelyben minden részecskének saját Kálmán

szűrője van. Az így előálló megoldás számítási igénye kisebb, mint a bővített Kálmán szűrők esetén.

Fontos még beszélni a gráf-alapú SLAM algoritmusokról is, amelyek képesek a részecske és bővített Kálmán szűrők hibáit orvosolni. Ezekben a módszerekben a kinyert adatokból gráfot építenek. A gráf csomópontokból és élekből áll, ahol az élek két egymást követő pozíció közti megkötéseket jelölik, amik lehetnek mozgások vagy mérések. A térkép előállításához, az összes élt linearizálják amelyből előáll egy ritka mátrix, ami a ritka gráf is egyben. Az optimalizálási folyamat miatt ez az algoritmus alkalmazása nagy léptékekben célszerűtlen.

Az ROS-ben elérhető SLAM algoritmusok:

HectorSLAM:

Gyakran használt csomag az ROS-ben. A szkennelt adatok összehasonlításán alapul, az eljárás neve „scan matching”. Az adatok összehasonlítását Gauss-Newton algoritmussal oldják meg. Az algoritmus megpróbálja megkeresni az alakjelző pontok optimális elhelyezkedését a már létrehozott térképhez. Az összehasonlítás során csak a fő alakjelző pontokat veszi figyelembe. A nagy előnye ennek az eljárásnak, hogy nincs szükség odometriára. A fejlesztők szerint ez megkönnyíti az alkalmazást terepi és repülő robotok számára. Viszont lassú távmérés és odometria hiánya esetén nagyon nagy pontatlanság léphet fel az adatösszehasonlításoknál, ez nyilván a létrejövő térképen is jelentkezik. A HectorSLAM algoritmust dolgozatom szoftveres részénél mélyebben is ismertetem.

Gmapping:

A leggyakrabban és legszéleskörűbben használt SLAM csomag az ROS-ben. A SLAM implementálása Rao-Blackwell részecske szűrőkkel történik. A bemenő adatai az odometria és a lézeres távmérő adatai. A kimeneti adat egy raszteres térkép, az akadályokról és a nyílt terepről, ábrázolja még a pozíciókat is. A legnagyobb előnye ennek a csomagnak a részletes és nagyméretű dokumentáció. A csomagot egyszerű konfigurálni és alkalmazni. Ez a csomag csak odometriával alkalmazható.

KartoSLAM:

Ez az algoritmus a Karto Robotics gráf alapú SLAM eljárásán alapul. Olyan algoritmus-optimalizációt sikerült elérniük, hogy a gráf alapú eljárásuk számítási igényben vetekszik a többi SLAM algoritmussal. A csomópontok az egymás utáni helyzeteket tartalmazzák sorban. Minden új csomópont esetén egy új optimális térbeli csomópont konfigurációt számol. A csomag alkalmazása nehézkes a hiányos dokumentáció és az odometria szükségessége miatt.

CoreSLAM:

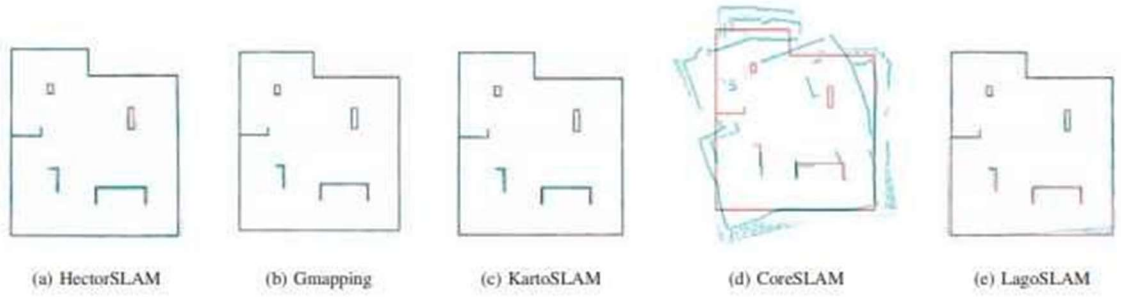
Egy egyszerű részecske szűrőt alkalmazva illeszti a szkennelt adatokat. A csomagnak szinte semmilyen dokumentációja nem létezik, így alkalmazása nagy nehézségekbe ütközik.

LagoSLAM:

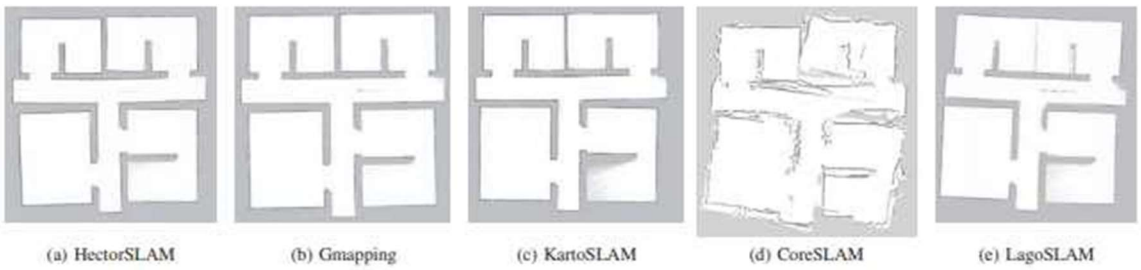
Gráf-alapú SLAM algoritmus; a többi gráf alapú eljárástól az különbözteti meg, hogy az optimalizációs eljárás során nincs szükség kezdeti becslésre. Ennek ára, hogy az a számítás komplexitását nagyban növeli. Az algoritmusról nincs semmiféle információ a hivatalos ROS oldalon.

CRSM SLAM:

Ennél az algoritmusnál az illesztést, egy véletlenszerűen frissülő mászó algoritmus hajtja végre. A térkép frissítése plusz dinamikus intenzitás információkkal történik. Ritkán használt módszer, az újabb ROS verziókban már nem támogatott.



3. ábra Eltéréstérképek szimulációs környezetben (Santos et al. - 2013)



4. ábra Eltéréstérképek valós környezetben (Santos et al. - 2013)

HectorSLAM algoritmus matematikája

A HectorSLAM egy nyílt forráskódú 2D SLAM technika. A módszer a lézerszkenner adatait veszi alapul, ami a környezetéből egy négyzetrács (grid) alapú térképet generál. A legtöbb grid alapú SLAM eljárással ellentétben nincs szüksége kerék-odometriára. Így a platform pozícióját csupán a mérési eredmények illesztéséből becsli. A modern LiDAR magas

frissítési frekvenciáját és pontosságát kihasználva gyors és pontos becsléseket végezhetünk a platform pozíciójára. (Kamarulzaman Kamarudin et al - 2014)

A mérési eredmények illesztési algoritmusa a Gauss-Newton-i megközelítésen alapul. Az algoritmus megpróbálja megtalálni a lézer végpontján lévő pontok pozícióját az előállított térképen úgy, hogy megkeresi a $\xi = (p_x, p_y, \psi)^T$ transzformáció hol minimális.

$$\xi^* = \arg\min_{\xi} \sum_{i=1}^{n} \left[1 - M\left(\mathrm{S}_i\left(\xi\right)\right)\right]^2$$

Ahol az $M(S_i(\xi))$ függvény visszaadja a térképi értéket $S_i(\xi)$-nél, ami a lézer végpontjának globális koordinátája. ξ kezdeti becslésével a léptetést Δ ξ becsülhetjük a hibamérték optimalizálásával:

$$\sum_{i=1}^{n} \left[1 - M\left(\mathrm{S}_i\left(\xi + \Delta\xi\right)\right)\right]^2 \rightarrow 0$$

Az $M(Si(\xi+ \Delta \xi)$ elsőfokú Taylor-sorba fejtésével és a Δ ξ szerinti parciális deriváltat nullával egyenlővé téve a Gauss-Newton egyenletet kapjuk a minimalizálási problémára:

$$\Delta\xi = \mathrm{H}^{-1} \sum_{i=1}^{n} \left[\nabla M(\mathrm{S}_i\left(\xi\right)) \frac{\partial \mathrm{S}_i\left(\xi\right)}{\partial \xi}\right]^T \left[1 - M\left(\mathrm{S}_i\left(\xi\right)\right)\right]$$

ahol:

$$\mathrm{H} = \left[\nabla M(\mathrm{S}_i\left(\xi\right)) \frac{\partial \mathrm{S}_i\left(\xi\right)}{\partial \xi}\right]^T \left[\nabla M(\mathrm{S}_i\left(\xi\right)) \frac{\partial \mathrm{S}_i\left(\xi\right)}{\partial \xi}\right]$$

Bár a HectorSLAM-ben nincs lehetőség explicit hurokzárásra (vagyis a mérés kezdeti helyére való visszazárásra, ezzel az illesztések pontosítására), a fejlesztők azt állítják, hogy a rendszer képes volt hurokzárásra több valós térképezési helyzetben is. A HectorSLAM algoritmus két nagy előnye a gyorsaság és az alacsony számítási igény.

A HectorSLAM egy Grid alapú SLAM, a végeredmény nem egy pontfelhő, hanem egy foglaltsági rács, ahol a raszter értékei -1-től 100-ig vehetnek fel értéket. -1 esetén ismeretlen, 0 esetén nincs akadály, 100 esetén az adott rácsban akadály van.

5. Platform hardveres megvalósítása

A platform alapját egy egyszerű barkácsüzletben is megvásárolható hobbi RC autó építéséhez használható váz adja, ez két DC motorból és egy előre lyukakkal ellátott plexi lapból áll.

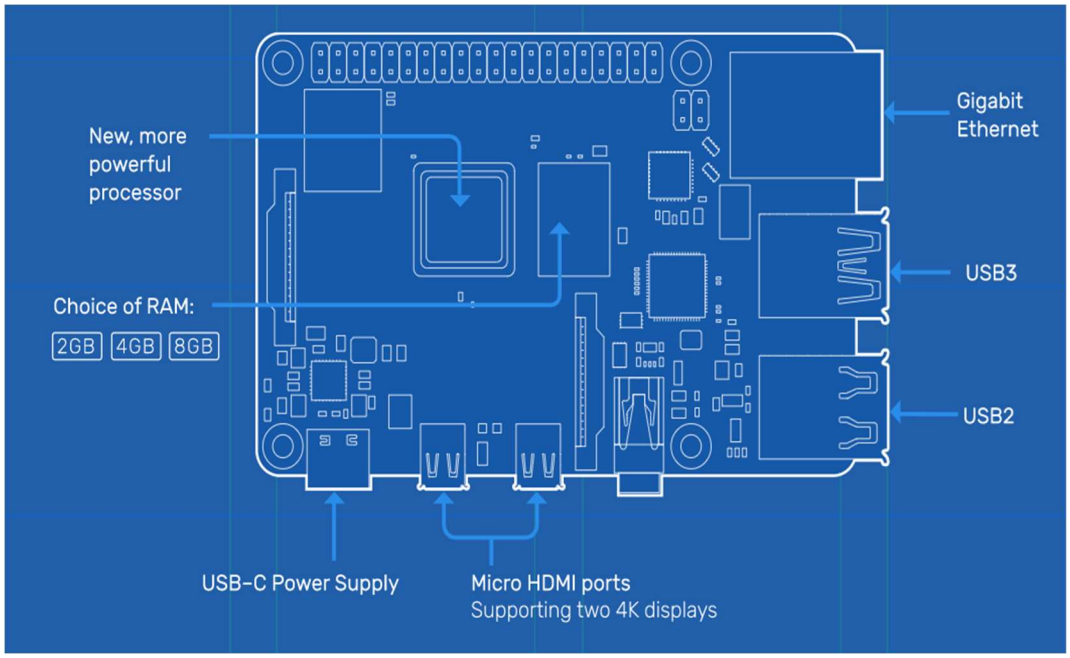
5. ábra A platform váza (https://www.amazon.de)

A platform számítási feladatait, a szoftverek futtatását egy Raspberry Pi 4 model B végzi, ennek is a 4GB memóriával felszerelt változata.

A Raspberry Pi egy bankkártya méretű, egyetlen áramköri lapra/kártyára integrált BCM2835 alapú egykártyás számítógép, amelyet az Egyesült Királyságban fejlesztettek oktatási célokra. A gép különböző Linux-disztribúciókkal működtethető, illetve elérhető egy RISC OS verzió is.

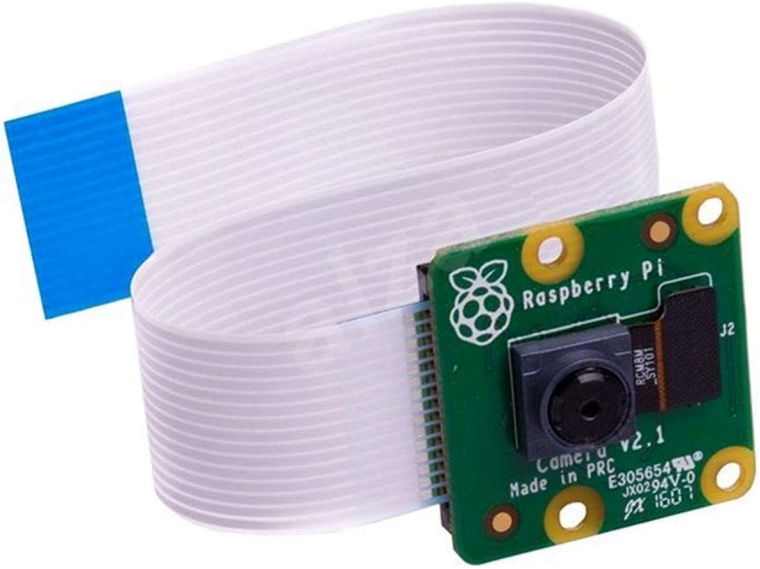
6. ábra Egy Raspberry Pi 4 model B (https://malnapc.hu/)

A Raspberry egy négy magos Cortex-A72 processzorral és 4 GB LPDDR4 memóriával van felszerelve. Megemlítendő még a lapkán lévő két darab USB2.0 és két darab USB3.0 csatlakozó. Megjelenítéshez két darab micro-HDMI porttal van ellátva. A tápellátása pedig egy USB-C csatlakozón, vagy a GPIO tűkön keresztül megoldható.



7. ábra A Raspberry Pi portjai (https://www.raspberrypi.org/)

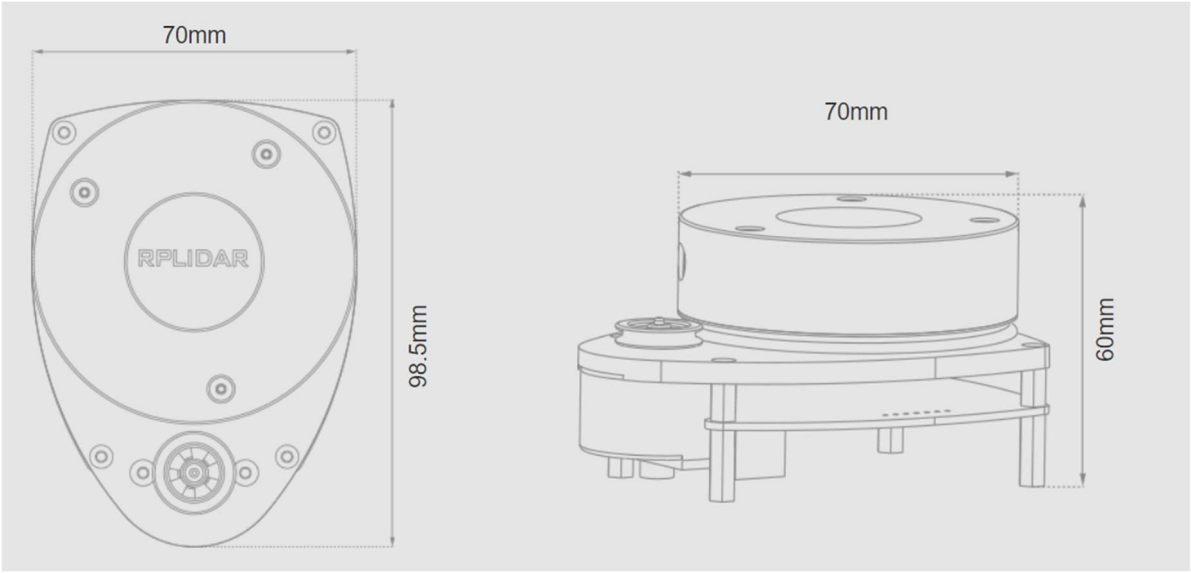
A platform elején egy konzollal rögzítve van a Raspberry-hez külön vásárolható Raspberry kameramodul is. A kameramodul egy 8 MP-s sony szenzorral van felszerelve. Négy darab egyszerű csavarral rögzíthető. Szalagkábellel csatlakozik a Raspberryhez.



8. ábra Raspberry Pi kameramodul (https://malnapc.hu/)

A lézerszkennelést egy a Slamtec cég által gyártott 2D lézerszkenner, az RPlidar A1-es végzi.

9. ábra Slamtec RPlidar A1 (https://www.slamtec.com)

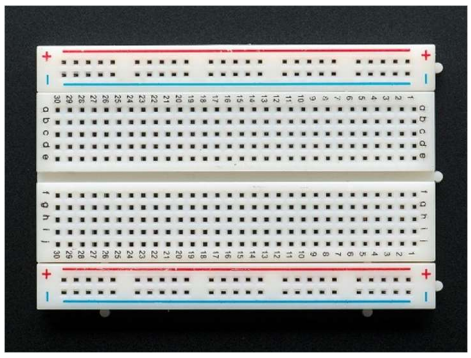
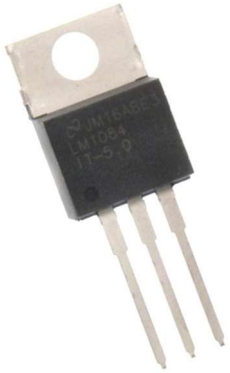
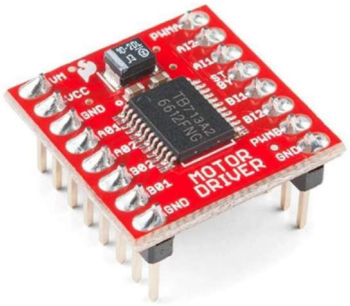


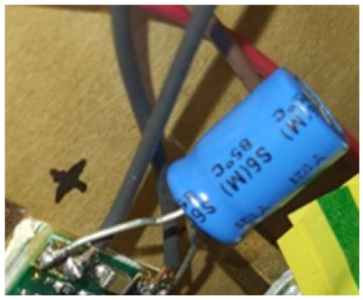
10. ábra Fizikai méretek (https://www.slamtec.com)

Az RPlidar A1 specifikációi:

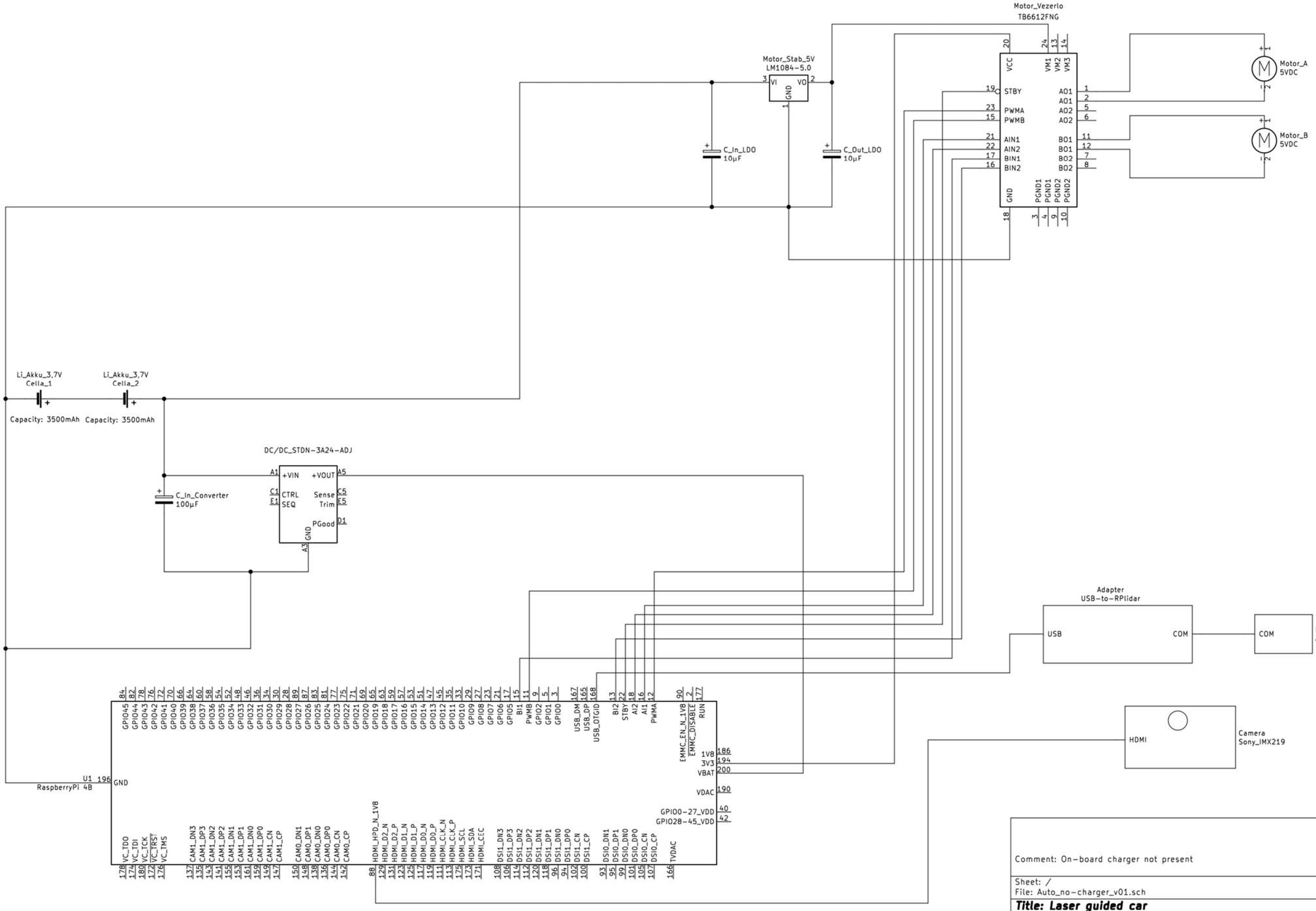
Hatótávolság:	0.15-12 m
Mérési tartomány:	360°
Mérés felbontása:	1,5 méteren belül: < 0.5 mm
	A teljes hatótávra: < a valódi távolság 1%
Szögfelbontás:	<1°
Egy méréshez szükséges idő:	0.5 ms
Mérési frekvencia:	minimum 2000 Hz | maximum 8000 Hz | átlagosan
≥4000 Szkennelési frekvencia:	minimum 5 Hz | maximum 10 Hz | átlagosan 5.5 Hz

A platform működéséhez szükséges egyéb elektronikai alkatrészek:

1 db egyszerű próbapanel, az áramkör kialakításához	
LM1084 lineáris feszültségszabályozó, a motorok áramköréhez	
TB6612FNG motorvezérlő modul, a motorok vezérléshez	
2 db Samsung 18650-3E 3500 mAh Li akkumulátor	

DSN-1504-3A Kapcsolóüzemű step-down feszültségszabályzó, állítható kimeneti feszültség szinttel	
Vezetékek a próbapanelhez és a Raspberry Pi GPIO tűs kimenetéhez.	
2 db 10 µF elektrolit kondenzátor	
1 db 100 µF elektrolit kondenzátor	
1db kétállású kapcsoló	

A következő oldalon látható a platform részletes kapcsolási rajza A3-as méretben. Az áramkör megtervezésénél és a számításokban Sárdy Balázs villamosmérnök volt a segítségemre.



Comment: On-board charger not present

Sheet: /
File: Auto_no-charger_v01.sch

Title: Laser guided car

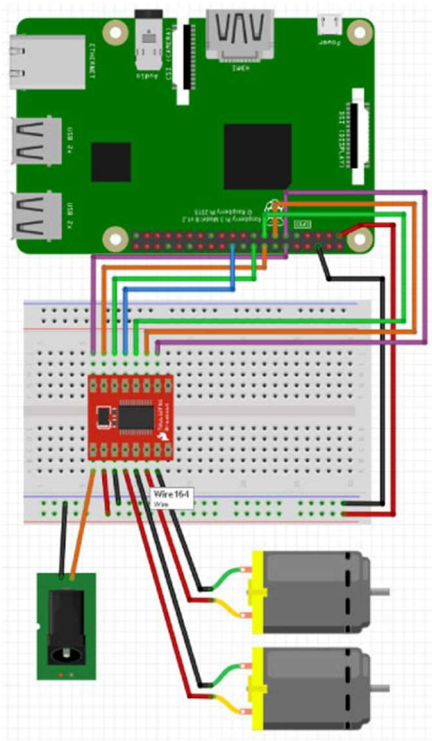
Size: A3	Date: 2020-10-19	**Rev: 003**
KiCad E.D.A. eeschema (5.1.7)-1		Id: 1/1

A platform áramkörének tervezésénél a fő szempont az volt, hogy a Raspberry Pi-hez megfelelő áramerősséget juttassunk megfelelő feszültséggel. A Raspberry 5V-os feszültség alatt nem fogja elindítani az operációs rendszert, 5V felett pedig károsulhat, akár javíthatatlanul tönkre is mehet. A stabil 5V biztosításához a kapcsolóüzemű step-down feszültségszabályzóra volt szükség. Ennél egy potenciométer segítségével állítható a kimenő feszültség. A motornál is szükség volt egy ilyenre, ott azonban elegendő volt egy lineáris feszültségszabályozó is, előre beállított kimenettel.

Mint a fenti rajzon is látható, egy 3,3V és egy 5V-os áramkör kialakítására is szükség volt. Az 5V-os a Raspberry tápellátását szolgálja, míg a 3,3V-osra a motorvezérlőnek van szüksége. A Raspberry Pi egy kapcsoló segítségével le is kapcsolható az áramkörről, így nemcsak feltöltött akkumulátorral lehet programozni. Ilyenkor a gyári tápegység USB-C csatlakozójával biztosítható a tápellátása.

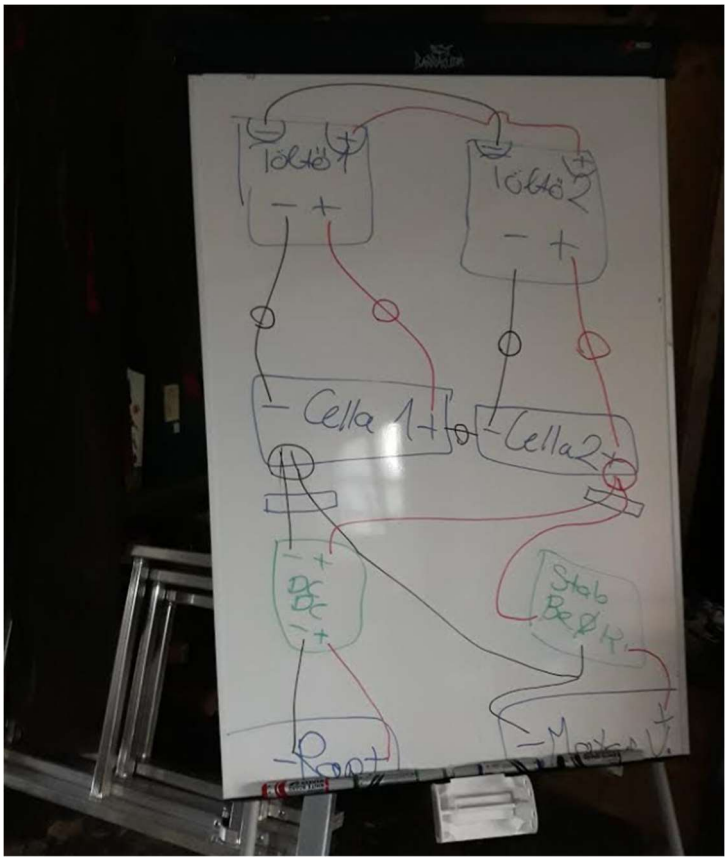
A lézerszkenner USB-n keresztül kapja az tápellátást, és ezen keresztül közvetíti a mért adatokat is. A rendszer háttértárja egy UHS Class 3, 32GB-os microSD kártya. A Raspberry kamera párhuzamos kábelen keresztül csatlakozik a Pi saját bemenetére. A Raspberry GPIO-n keresztül kapja a tápellátást, ha a kapcsoló felfelé van kapcsolva, különben csak az USB-C-n keresztül működtethető.

A motorvezérlő részletes bekötése a motorra és a Raspberry GPIO-jára az alábbi ábrán látható részletesen:



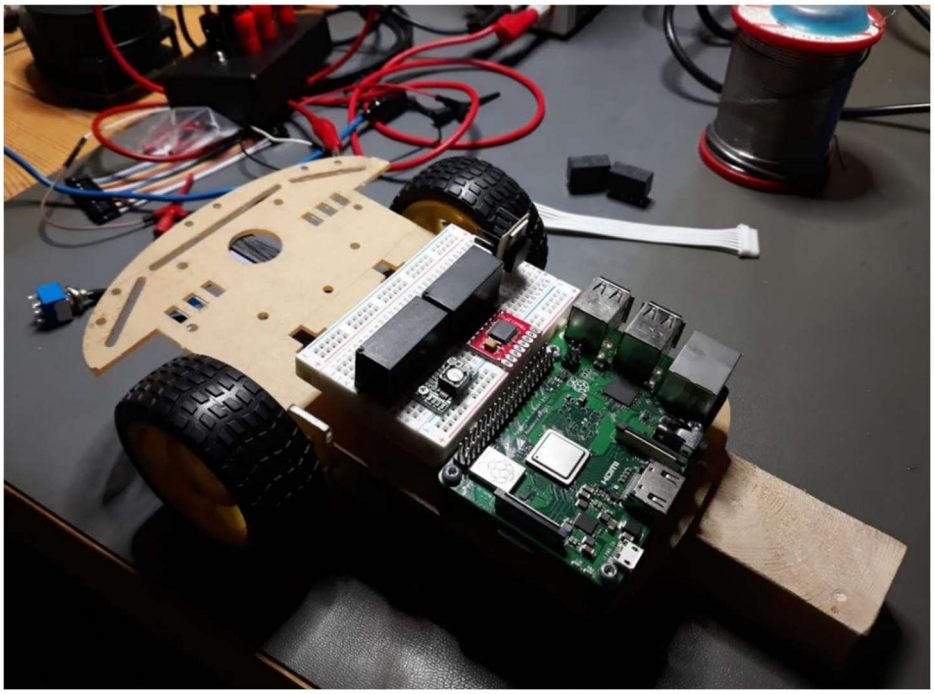
TB6612FNG	
VM	EXT Power (V)
GND	EXT Power (GND)
GND	Raspi (GND)
VCC	Raspi (3.3v)
A01	MotorA (+)
A02	MotorA (-)
B01	MotorB (+)
B02	MotorB (-)
PWMA	Raspi (Pin 12)
AI1	Raspi (Pin 16)
AI2	Raspi (Pin 18)
STBY	Raspi (Pin 22)
BI1	Raspi (Pin 15)
BI2	Raspi (Pin 13)
PWMB	Raspi (Pin 11)

11. ábra Raspberry GPIO, és motorvezérlő bekötése



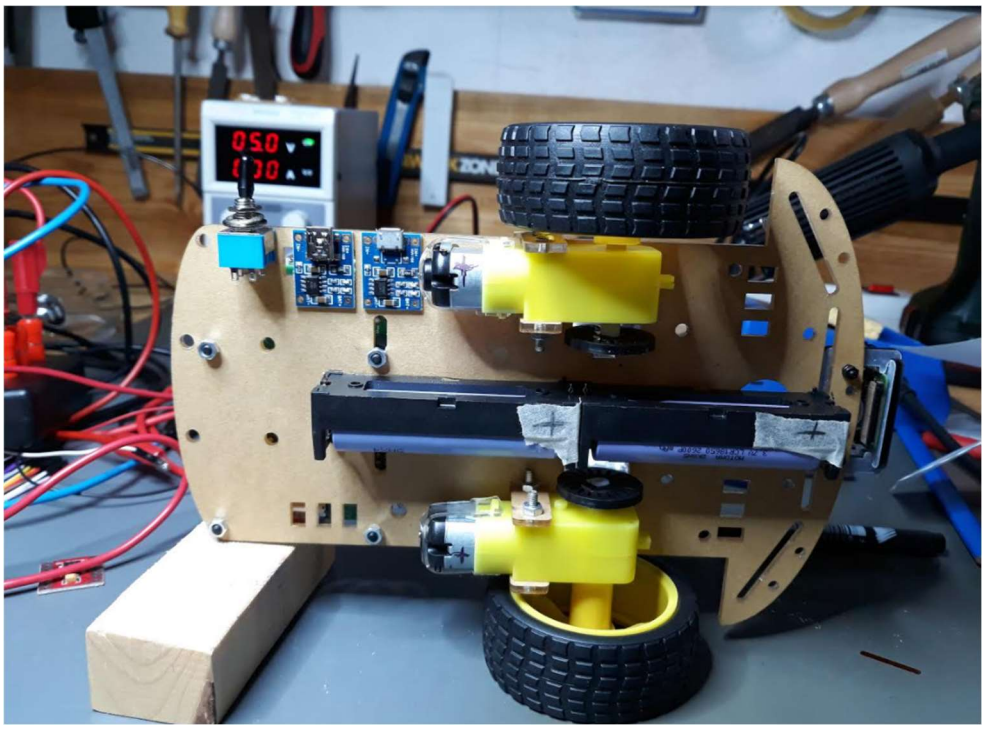
12. ábra Koncepció és kapcsolási rajz

A vázhoz először a Raspberry Pi-t és a próbapanelt rögzítettük. A Raspberry nyáklapján ehhez előre fúrt csavarhelyek találhatók. Hogy a nyáklap alatt maradjon hely, távtartókat alkalmaztam.



13. ábra A platform építésének első lépései

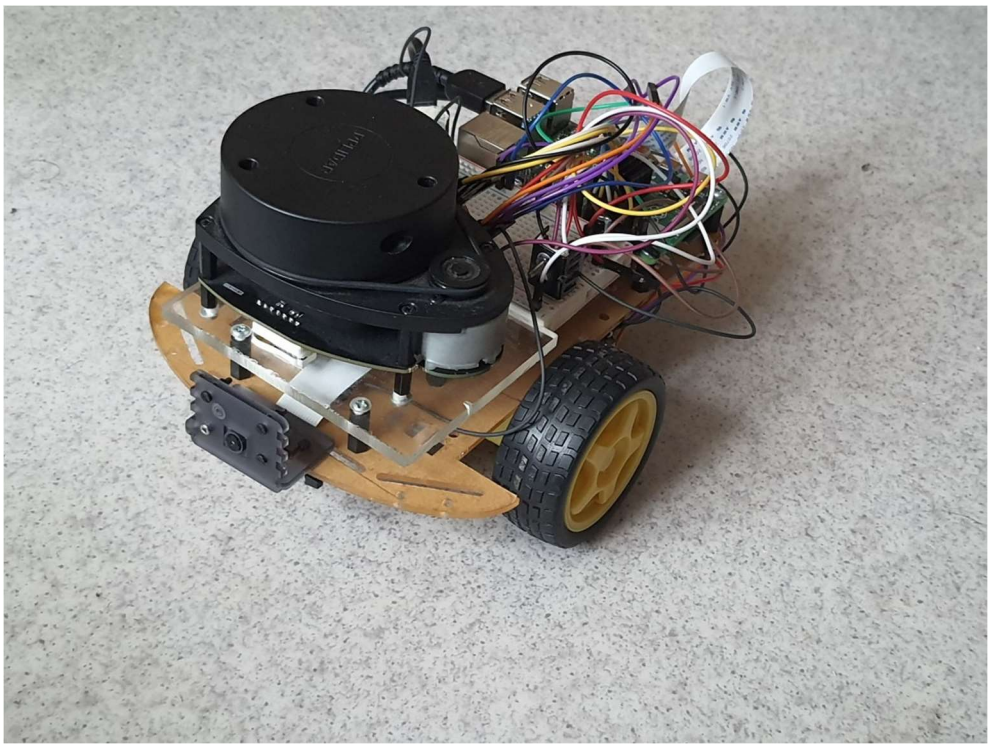
Az akkumulátorok foglalata a váz alsó felén kapott helyet a két motor közt.



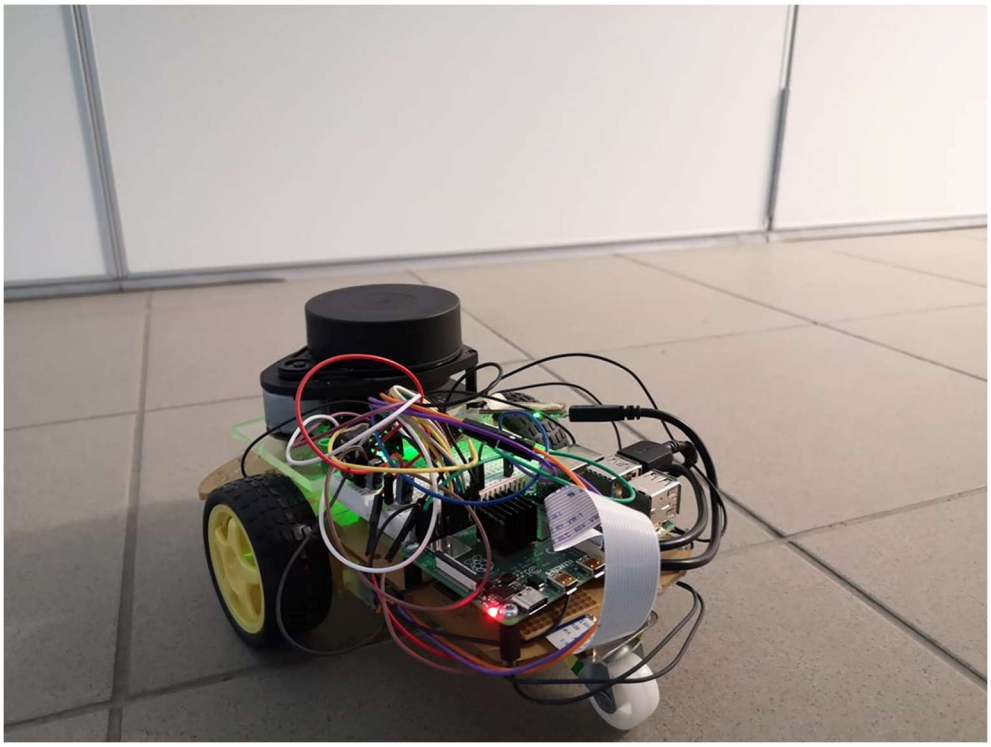
14. ábra Az akkumulátorok

A lézerszkenner a platform elején emelt pozícióban került elhelyezésre, így a Raspberry és a kábelek nem takarnak ki semmit a 360°-os mérési tartományból. A megemelést egy kisebb plexilappal és távtartókkal valósítottam meg.

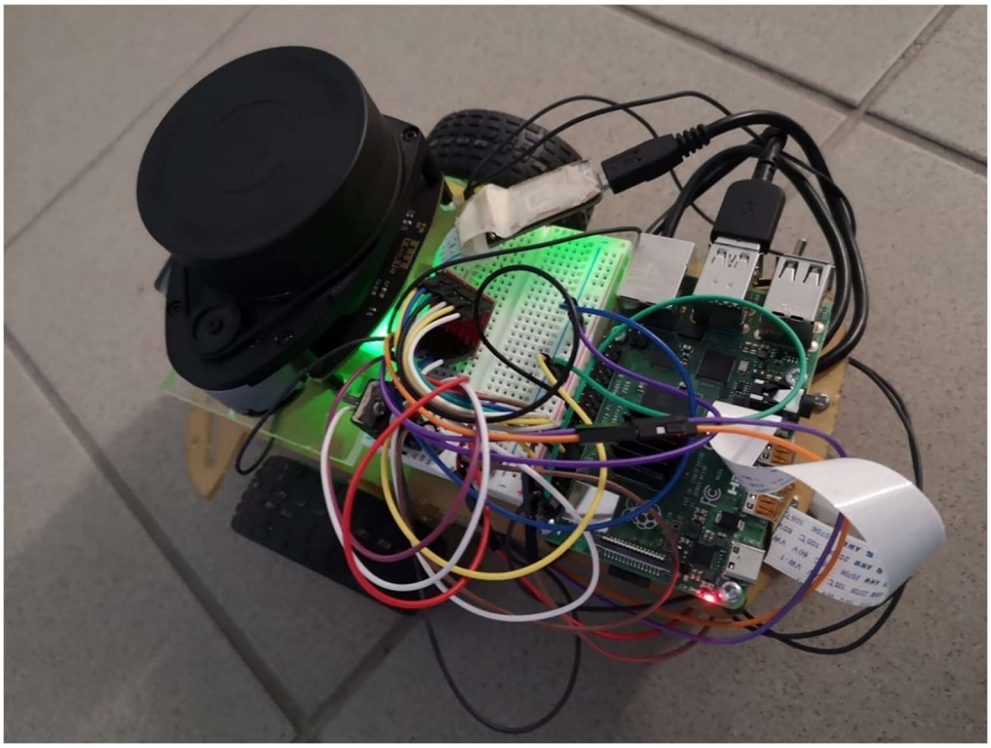
A platform végül a következőképpen állt össze:



15. ábra A végleges platform szemből



16. ábra A végleges platform hátulról



17. ábra A végleges platform felülről

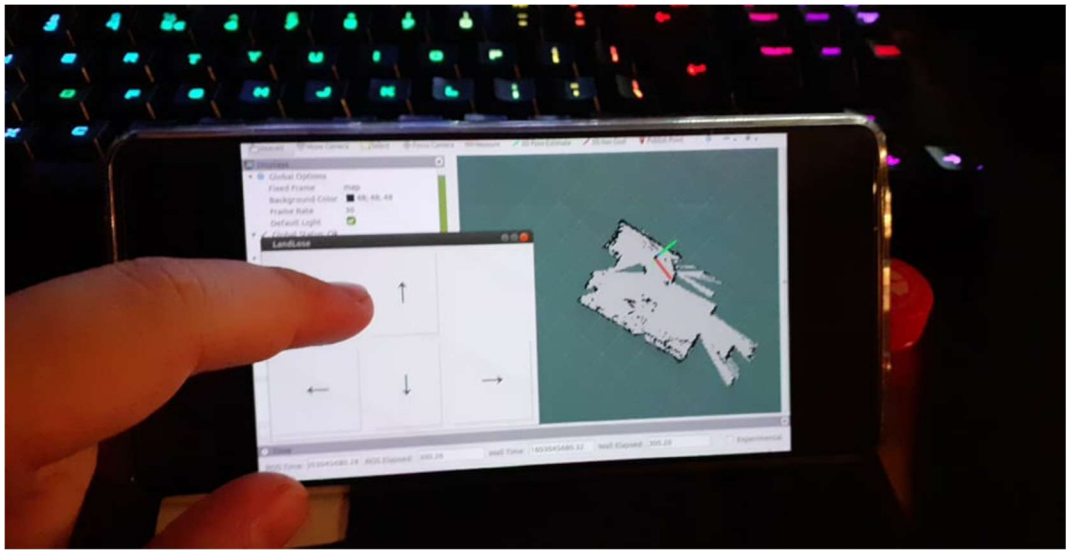
6. Szoftveres megvalósítás

A rendszer operációs rendszerének az Ubuntu MATE legújabb verzióját, a 20.04-es Focal Fossa kódnevűt választottam. Ez szükséges az ROS legújabb Noetic Ninjemys kódnevű verziójához. Az Ubuntu MATE egy ingyenes és nyílt forráskódú Linux disztribúció, az Ubuntu mobileszközökre szánt változata. Főként a felhasználói felületben tér csak el a klasszikus Ubuntutól. A Raspberry Pi-hez a gyártók által ajánlott operációs rendszer a Raspbian, ami egy Debian alapú disztribúció. A fő oka annak, hogy az Ubuntu mellett döntöttem az volt, hogy az ROS támogatottsága sokkal jobb Ubuntu-n, mint Debian-on.

Az Ubuntu-t lehetett volna „headless”-ként is konfigurálni, ami azt jelenti, hogy nem lenne grafikus felhasználói felület az operációs rendszerhez. A programozás és az irányítás megkönnyítése miatt döntöttem a grafikus felület mellett. A rendszert úgy konfiguráltam, hogy akkor is legyen grafikus felület, ha nincs csatlakoztatva kijelző hozzá. Ezen felül a rendszer induláskor, a bootolás után egyből elindítja a VNC szervert. A VNC szerver-alkalmazás biztosítja, hogy a rendszer távolról vezérelhető legyen. A VNC egy grafikus asztal-megosztási alkalmazás, ami az RFB protokollt használja. Átviszi a billentyűzeten és az egéren kiadott utasításokat, és közel valós időben pár tized másodperces késleltetéssel megjeleníti a grafikus felületet a másik számítógépen.

18. ábra A platformhoz csatlakozás VNC Viewer alkalmazással

A VNC-s megoldás segítségével WiFin keresztül vezetékmentesen vezérelhetem a Raspberryt. Nagy előnye, hogy okostelefonról is megoldható a vezérlés. A rendszerhez csatlakozáshoz csupán az szükséges, hogy egy Wifi hálózaton legyenek, és ismerjük a platform helyi IP-címét. Ez PC-vel és router segítségével egyszerűen megoldható, de Androidos rendszeren ezek az opcióink elég korlátozottak. Ha ismerjük a platform IP-címét, egyszerűen az 5900-as porton keresztül csatlakozhatunk a VNC szerverhez. Példa egy ilyen IP-címre: *192.168.137.112:5900*.



19. ábra Telefonról is csatlakozhatunk a rendszerhez, és vezérelhetjük is

Az ROS telepítéséhez szükséges néhány előkészítő lépést is elvégeznünk. Elsőként fel kell készíteni az Ubuntu alapú rendszert, hogy csomagokat telepítsen az ros.org-ról is. Ez a következő paranccsal tehető meg:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

Ezután be kell állítani a hozzáférési kulcsunkat:

```
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

Ezt követően az ROS-t 3-féle terjedelemben telepíthetjük:

Asztali teljes verzió: Minden az asztali verzióból, plusz 2/3D szimulátorok, arénák

```
sudo apt install ros-noetic-desktop-full
```

Asztali verzió: ROS alap, plusz grafikus megjelenítéshez eszközök

```
sudo apt install ros-noetic-desktop
```

ROS alap: (csak, ami a működéshez szükséges) ROS csomagkezelés, kommunikációs könyvtárak semmilyen grafikus megjelenítési eszköz

```
sudo apt install ros-noetic-ros-base
```

A telepítés után minden terminálban, ahol használni szeretnénk be kell állítani a környezetet

```
source /opt/ros/noetic/setup.bash
```

Miután telepítettük az ROS-t, létre kell hozni egy munkakönyvtárat. Esetemben ez catkin_ws névre hallgat. Ebbe a könyvtárba kell minden csomagot és node-ot bemásolni, amit használni szeretnénk.

Ha minden megtalálható a mappában, a `catkin_make` paranccsal le kell fordítanunk minden C++-ban megírt node-ot; enélkül ugyanis az ROS használhatatlan. Ezt minden egyes új node-nál és package-nél meg kell tenni. A rendszer figyelmeztet, ha valamelyik node-ot nem sikerülne fordítani.

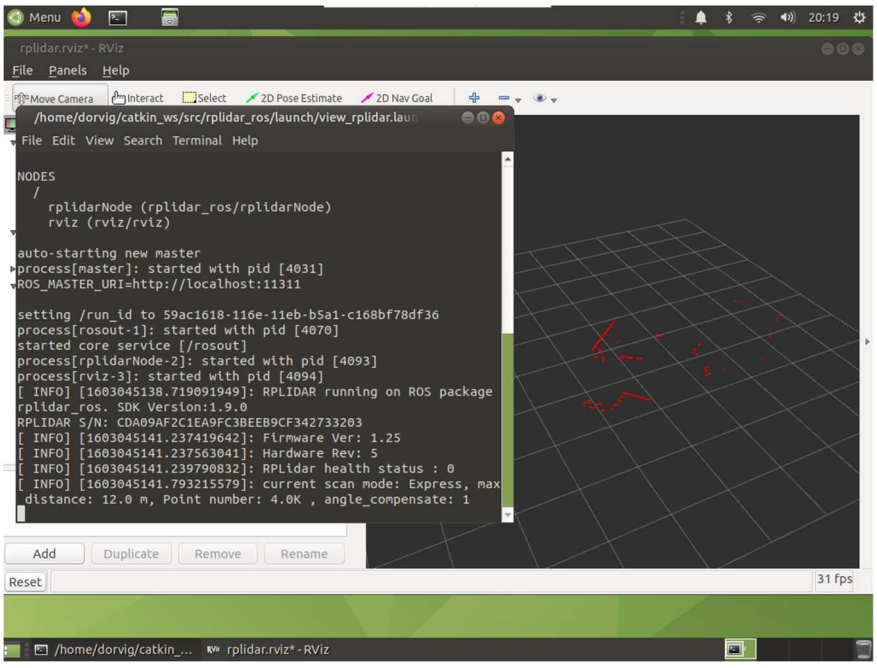
Ha a `catkin_make` parancs hiba nélkül lefut, a rendszer készen áll az ROS node-ok futtatására. Az ROS-es node-ok indítása parancssoron keresztül a roslaunch parancs, az adott node nevének, és a .launch fájl megadásával történik. A .launch fájlokat magunknak kell testreszabni, ezek igazából .xml formátumban a futtatáshoz szükséges összes argumentumot tartalmazzák.

```
<?xml version="1.0"?>

<launch>

  <arg name="geotiff_map_file_path" default="$(find hector_geotiff)/maps"/>

  <!-- <param name="/use_sim_time" value="true"/> -->
  <param name="/use_sim_time" value="false"/>

  <node pkg="rviz" type="rviz" name="rviz"
    args="-d $(find hector_slam_launch)/rviz_cfg/mapping_demo.rviz"/>

  <include file="$(find hector_mapping)/launch/mapping_tif.launch"/>

  <include file="$(find hector_geotiff)/launch/geotiff_mapper.launch">
    <arg name="trajectory_source_frame_name" value="scanmatcher_frame"/>
    <arg name="map_file_path" value="$(arg geotiff_map_file_path)"/>
  </include>

</launch>
```

20. ábra Példa egy .launch fájl tartalmára

1.a Lézerszkennelés és SLAM

A lézerszkenner adatainak gyűjtését az ROS Rplidar_node nevű csomagja végzi. A node-nak hozzáférést kell adni az USB porthoz, ahol az RPlidar csatlakoztatva van, ezután a mérési eredményeket az ROS-ben a beállított témán (topic) publikálja, esetemben ez a /scan topic. Erre a topicra feliratkozhat ezután bármelyik másik node is. A mérési eredményeket a .launch fájl megváltoztatásával valós időben meg is jeleníthetjük az rviz felületén.

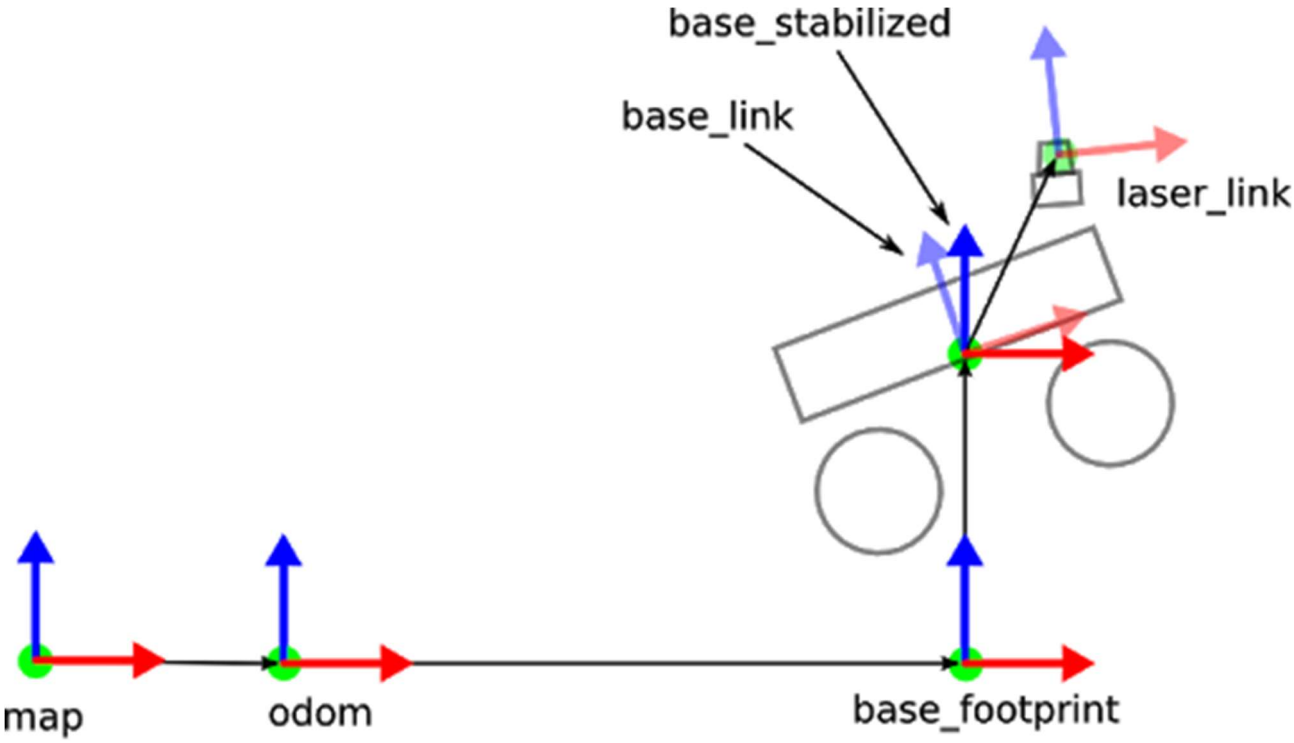


21. ábra Az RPlidar node futtatás közben,
az éppen mért pontok grafikus megjelenítésével

Ezután futtatható a hector_mapping node is. Ez a node feliratkozik a mérést végző node /scan topicjára, és ebből állítja elő a térképet. A.launch fájlban sok szükséges argumentumot meg kell adnunk, hogy megfelelően működjön a rendszer. A legfontosabb ezek közül a különböző koordinátarendszerek összehangolása, és az ezek közti transzformációs paraméterek megadása.

A transzformációkat a tf node végzi, a node neve egy rövidítés, eredetileg transformation tree. Ezzel generálhatunk ábrát is a különböző koordinátarendszerek kapcsolatáról.

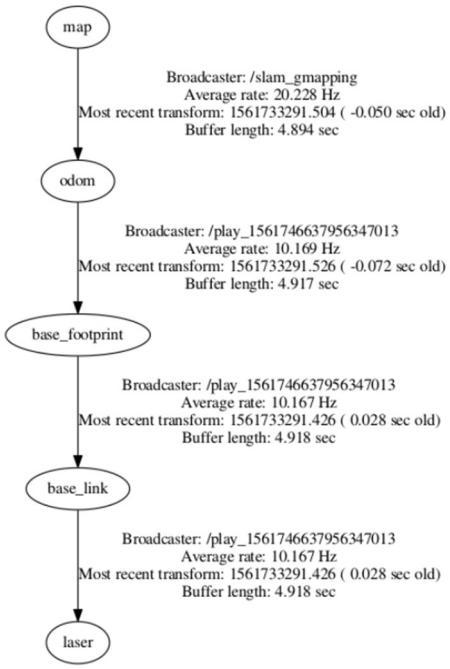
Érdemes először tárgyalni, milyen koordinátarendszerekről is beszélhetünk, egy egyszerű platform esetén a következő koordinátarendszerekről kell tudnunk (19. ábra).



22. ábra Koordinátarendszerek

Esetemben nagyon leegyszerűsödik ez a kép, a base_stabilized, és a base_footprint egyenlő lesz a base_linkkel. Amíg sík terepen maradunk, és nincs szükség a magasságra, ezek egyenlők. A laser_link a szkenner koordinátarendszere, ezt az indítófájlban egyenlővé tesszük a base_linkkel, vagyis a lézer szkenner rendszere megegyezik a platformával, s mivel az odometriát is a base_link rendszerből számolja vissza, így ez a két rendszer is egyenlővé tehető.

Ezután a transzformációs fa nagyban egyszerűsödik:



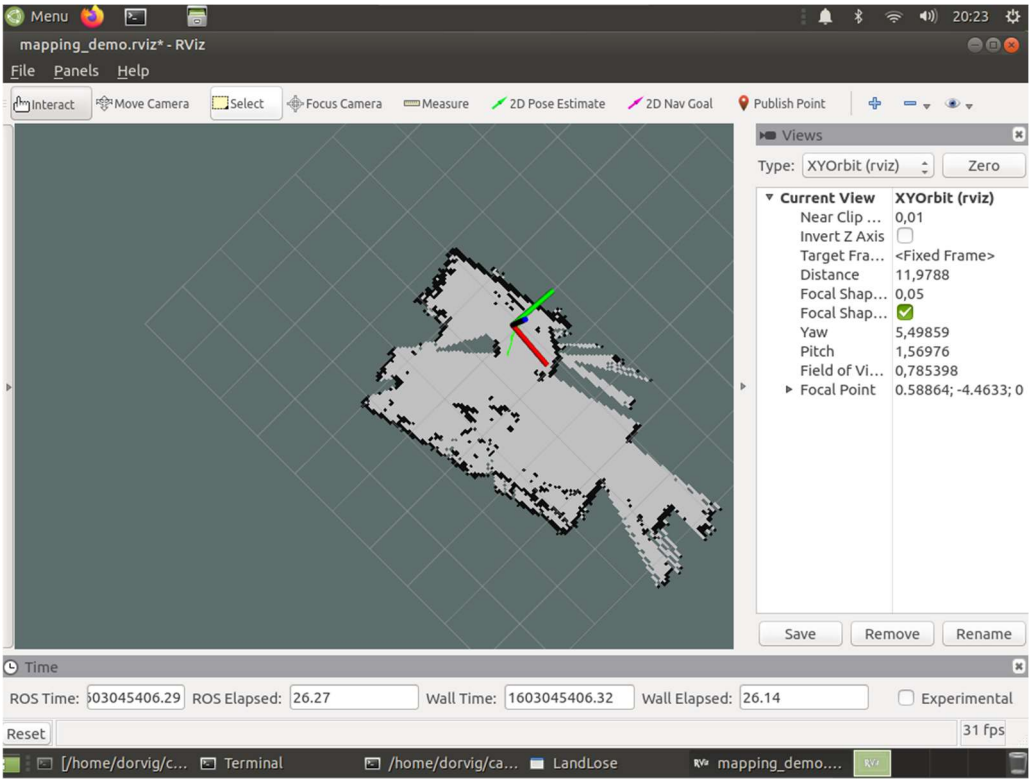
23. ábra Transzformációs fa

Az egyetlen, amit nem tudunk „egyenlővé” tenni, az a base_link és a laser_link, ezek között szükséges megadnunk egy transzformációt. Ez egy egyszerű transzformáció lesz, minden paraméterében 0. Ezt egy statikus transzformációként minden mérési eredményhez hozzá rendeljük, innentől az tf node ezt a /tf topicon publikálja.

Ezt a transzformációt a tf node static_transform_publisher folyamata végzi, a bemenő paraméterei:

- az X irányú eltolás
- az Y irányú eltolás
- az Z irányú eltolás
- yaw – forgatás Z tengely körül
- pitch – forgatás Y tengely körül
- roll – forgatás X tengely körül
- period_in_ms – milyen gyakran küldje a transzformációs üzeneteket 100ms (10Hz) általában jó érték

Erre a topicra a hector_mapping node-nak lesz szüksége. Ha sikeresen megírtuk az indítófájlt a node elindítása után, rviz-ben valós időben követhetjük a térkép létrejövetelét:

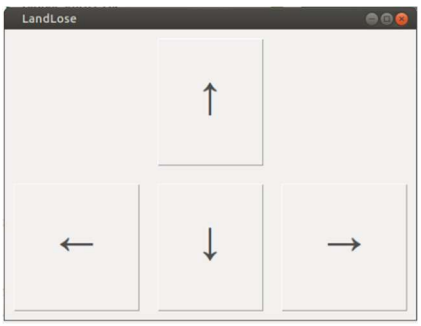


24. ábra A hector_mapping node munka közben

A fenti ábrán egy kollégiumi szobáról készült térkép látható az rviz megjelenítőben; körülbelül 5 másodpercnyi mozgatás után ilyen térképet kaphatunk. A szoba falai jól kivehetők, de a bútorok miatt erről a helyről nagyon sok a kitakarás.

b. Irányítás megvalósítása pythonban

A platform így már képes térképezésre, de még emberi erővel kell a mozgatást megoldani. Ennek megoldására pythonban írtam egy grafikus felülettel rendelkező programot, amivel irányítható a platform. A program felülete egyszerű és kézreálló. Négy irányító gombból áll: előre, hátra, jobbra, balra.



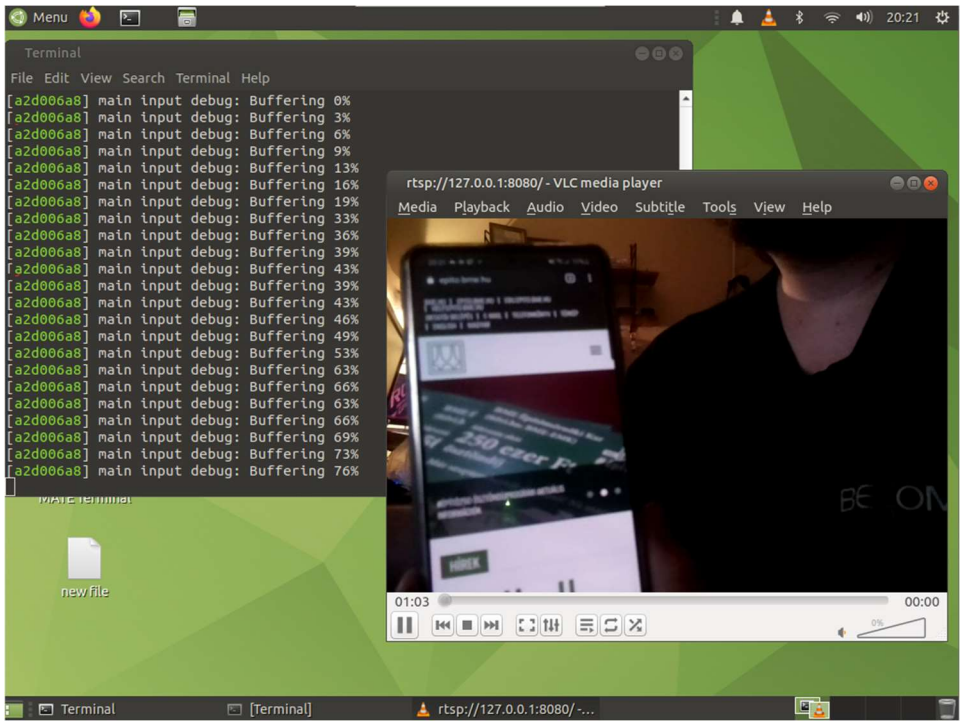
25. ábra Az irányítófelület

A program egyszerű alapokon nyugszik, néhány függvényből áll csak. Egyetlen bemenő adat hozzá a motorteljesítmény százalékosan, ez lesz a platform sebessége.

Ha megnyomjuk az előre gombot mindkét motornak jelet küld és amíg lenyomva tartjuk a gombot, mindkettőt előre, hátra gomb esetén ugyanez történik csak a másik irányba. Jobbra és balra fordulásnál pedig egyszerűen ellentétesen pörgeti a két kereket.

c. Kamera felhasználása az irányításhoz

A kamera vezérléséhez a Raspberry kamerához írt raspivid programot használtam fel. A program segítségével valós idejű videó-streamet indítok, majd a VLC videólejátszó segítségével rácsatlakozunk a streamre a localhost (127.0.0.1) beállított portján (az alábbi ábrán 8080-as) és már látható is a kamera képe. Ez azonban nagy késleltetéssel jár, érdemes lenne vizsgálni más VNC-s megoldásokat is.



26. ábra Kamera képének bemutatása,
háttérben a raspivid stream terminálja

d. Mérési eredmények kinyerése

A mérési eredményekből készített térkép exportálására eredetileg a hector_geotiff node szolgálna. Ezt sajnos nem sikerült futtatnom, valamilyen probléma miatt nem működik.

Ennek kiküszöbölésére kétféle eljárást találtam ki

- Az rviz megjelenítőben van lehetőség a térkép kimentésére .png formátumban,
- Az ROS-nek van egy rosbag parancsa, amivel a különböző topicokat menthetünk ki az ROS saját formátumában, ennek a segítségével bármikor szimulálhatjuk a mérés elvégzését.

Próbálkoztam a /map topic .txt-be mentésével is, ez másodpercenként 100MB adatot kreált, ami már elfogadhatatlan adatméret.

Burok szkriptfájlokat írtam a különböző node-okra és alkalmazásokra, hogy ne kelljen mindig parancssorból meghívni azokat. Ezek elindítása a szkriptfájlra kattintással történik. A szkriptfájljaim a következők:

- rplidar.sh – elindítja az lézerszkenneres mérést,
- slam.sh – elindítja a hector_mapping node-ot és valós időben figyelhető a készülő térkép,
- joystick.sh – elindítja a grafikus vezérlőfelületet,
- stream.sh – elindítja a kamera képéről a streamet,
- show.sh – VLC lejátszóban megmutatja a kamera képét,
- log.sh – a rosbag parancs bag formátumában elmenti az összes topicot amin publikáció történik.

Az rvizből kimentett képek a négyzetrácshálónak hála könnyedén georeferálhatók.

7. Mérési eredmények, alkalmazás

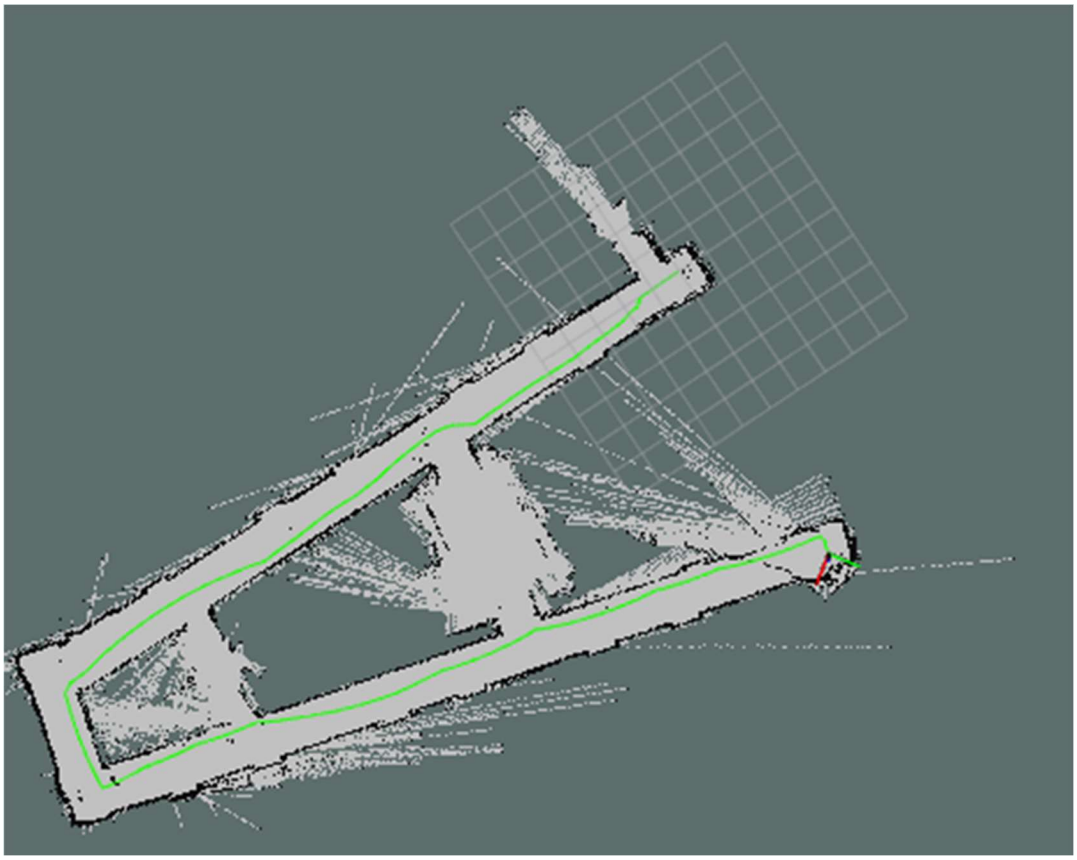
A rendszert négy helyen vettettem be éles mérésre.

Az első térképezést a Vásárhelyi Pál Kollégium (VPK) 10. emeletének folyosóin végeztem. Ennél a mérésnél sajnos nem tudtam összehasonlítási alapot beszerezni. Az összehasonlítás alapjául a folyosó faláról fényképezett kiürítési terv szolgált.

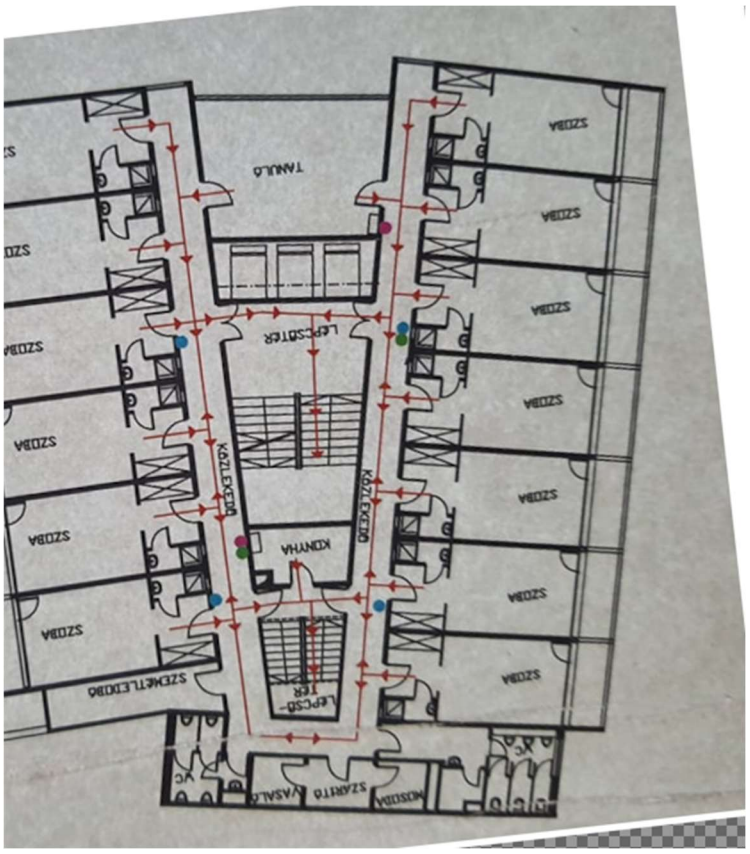
Az alábbi térkép elkészítéséhez háromszor kellett a platformot végig vezetnem a folyosókon. Az első két vezetésnél a motorteljesítmény túl magas volt, így a térkép illesztése nem sikerült. A sebességet a harmadik alkalommal empirikus úton állapítottam meg.

A rendszer nagy gyengesége volt még az első alkalmazásnál, hogy a csapágyas hátsó kerék szabadon forgott. Az illesztési algoritmus nagyon érzékeny a forgatásokra, főleg a nagysebességű forgatásokra. A hátsó szabad kerék miatt azonban nagyon sokszor kellett korrigálni a pályát kis fordulásokkal. Így a hátsó szabad kereket ragasztószalag segítségével rögzítettem, ezután a platform könnyebben vezethetővé vált, és az illesztés is sokkal megbízhatóbb lett.

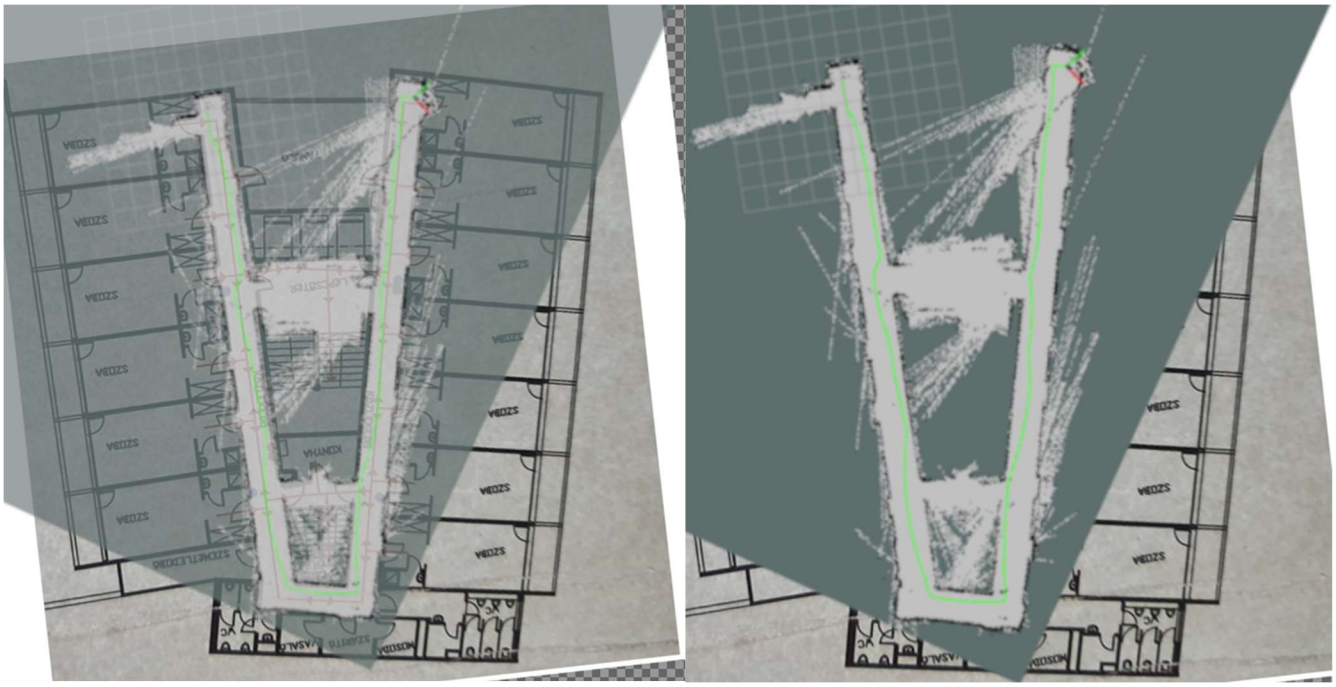
A kollégiumról készített térkép idején még szabadon volt a kerék; a térkép jobb alsó sarkában látszik is, hogy a hirtelen fordulás miatt az egész folyosó hirtelen elcsavarodott kb. 90 fokkal. A térképen zöld vonallal látszik a platform által bejárt út is.



27. ábra VPK 10. emelet

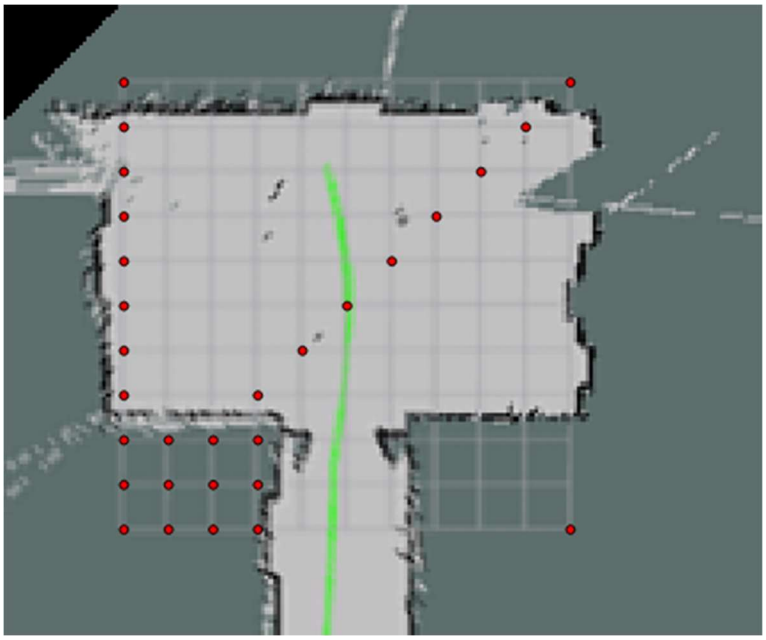


28. ábra A kiürítési tervről készített fénykép

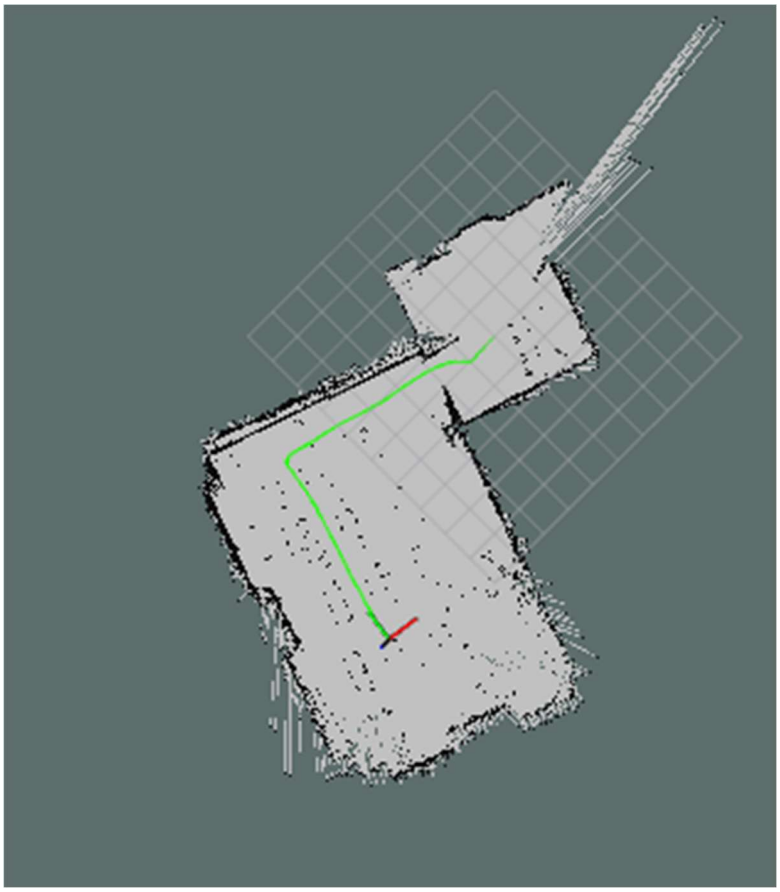


29. ábra A készített térkép összehasonlítása a kiürítési terv fényképével

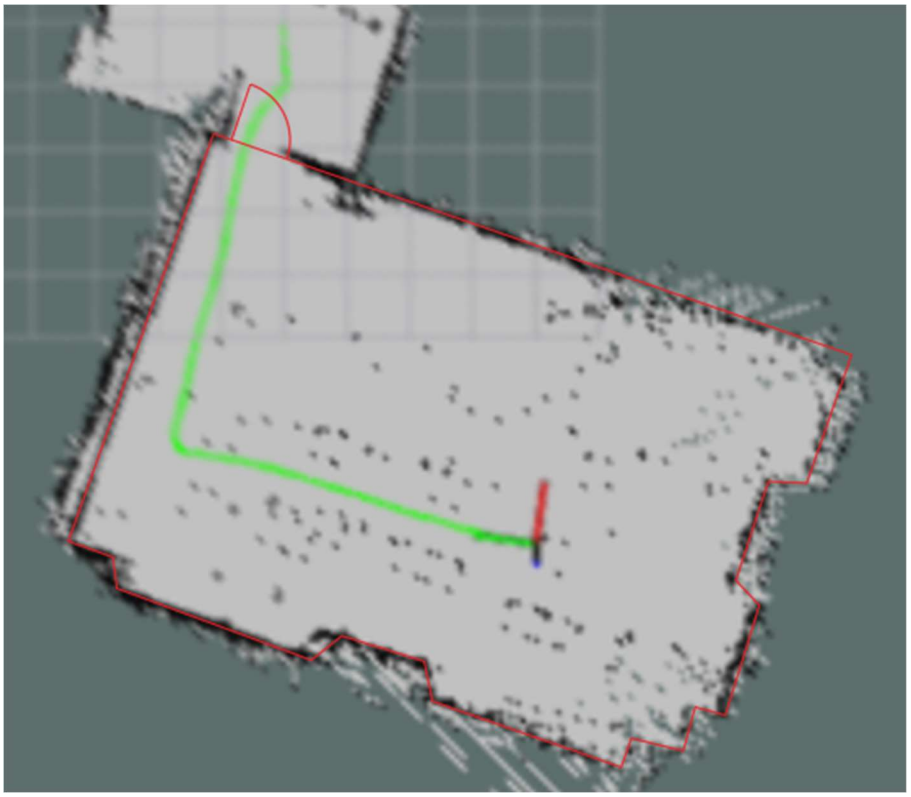
A következő térképezést az Általános- és Felsőgeodézia Tanszék Rédey-termében végeztem. A K épület alsó szintjéről sikerült beszereznem egy .dwg alaprajzot, de ez még a tanszék felújítása előtt készült, így nem szolgálhat pontos összehasonlítási alapként. A térkép georeferálása alapján Autocadben megrajzolható a terem alaprajza.



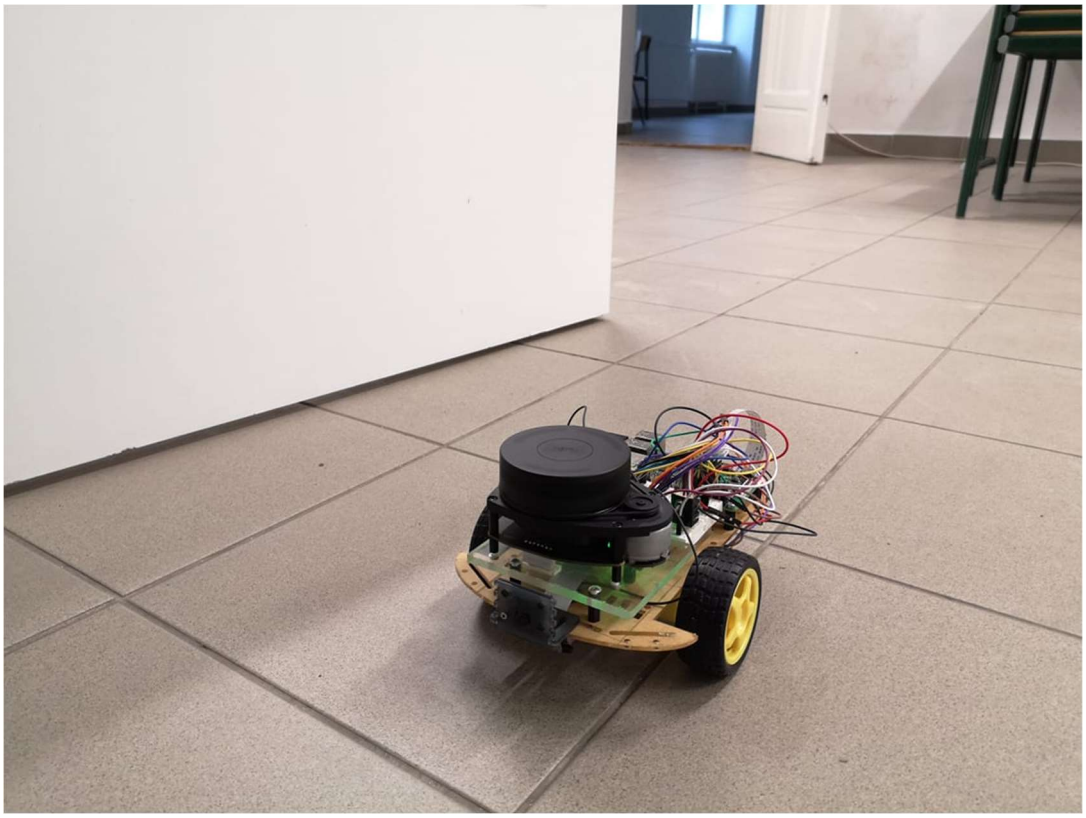
30. ábra A georeferálás QGIS-ben történt, a méteres rácsháló alapján



31. ábra A Rédey teremről előállított térkép, zölddel a platform által bejárt út



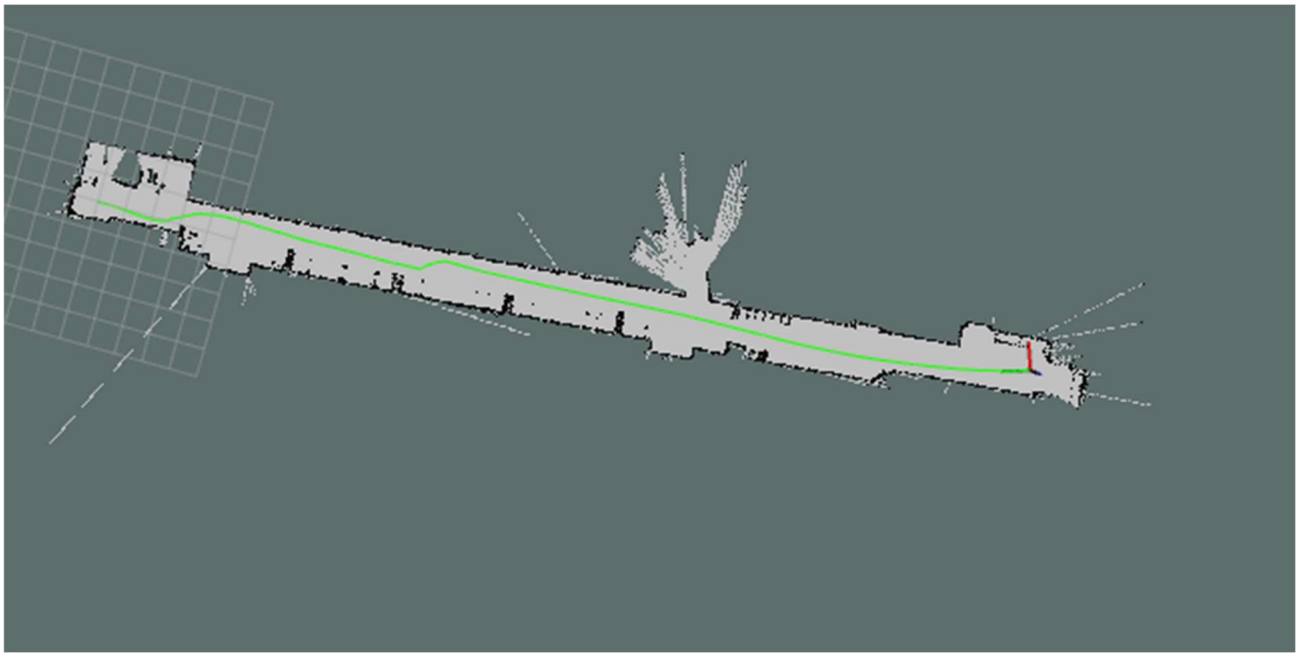
32. ábra Rédey terem alaprajza



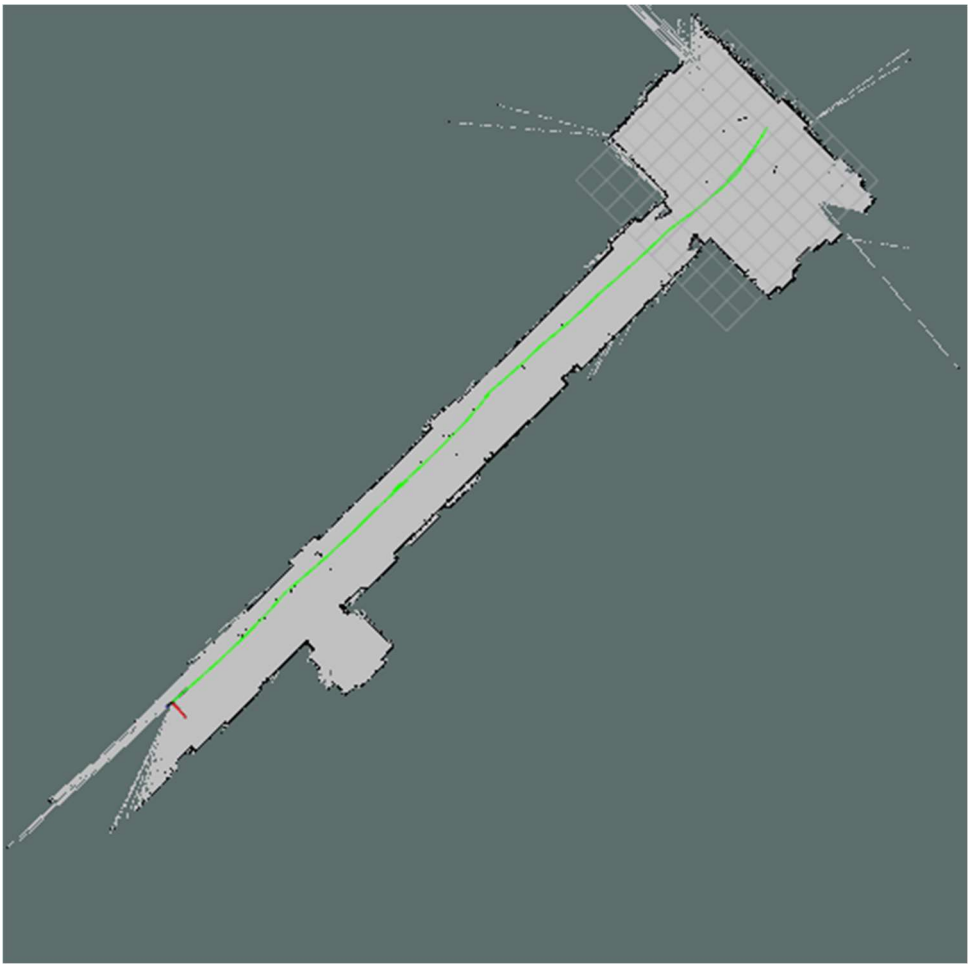
33. ábra A platform térképezés közben

34. ábra A platform vezérlése okostelefonnal történt

Az utolsó két helyszín az Fotogrammetria és Térinformatika Tanszék folyosója és a tanszék bejárata előtt található folyosószakasz volt.

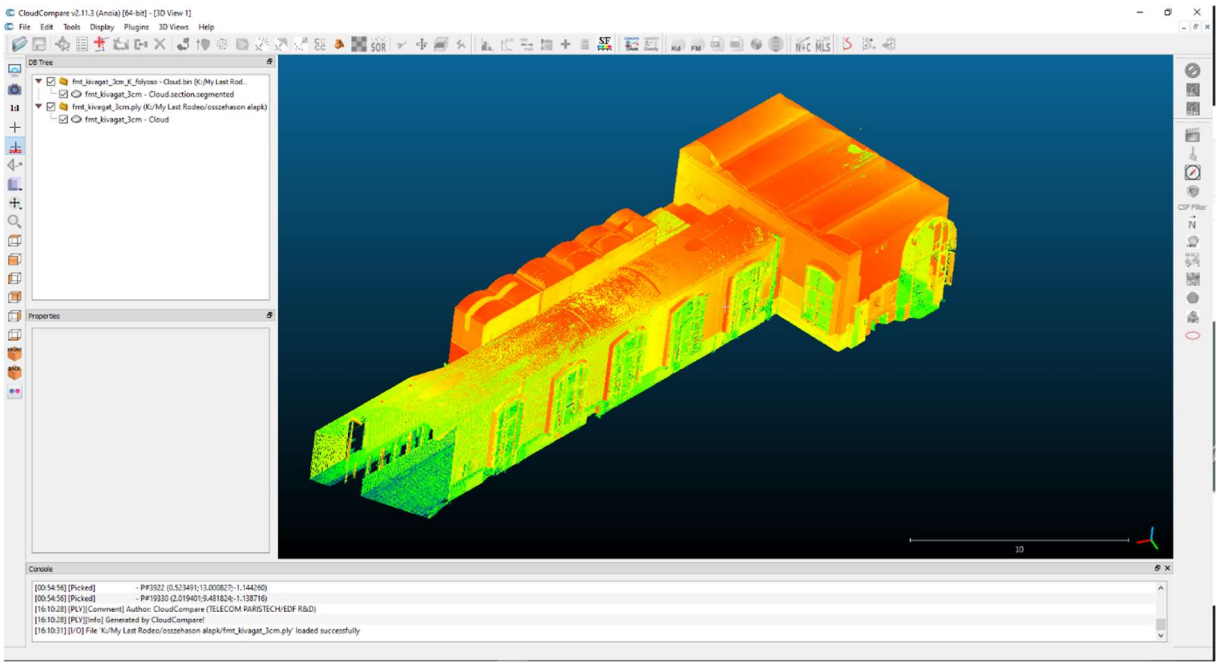


35. ábra Az FMT belső folyosójáról előállt térkép



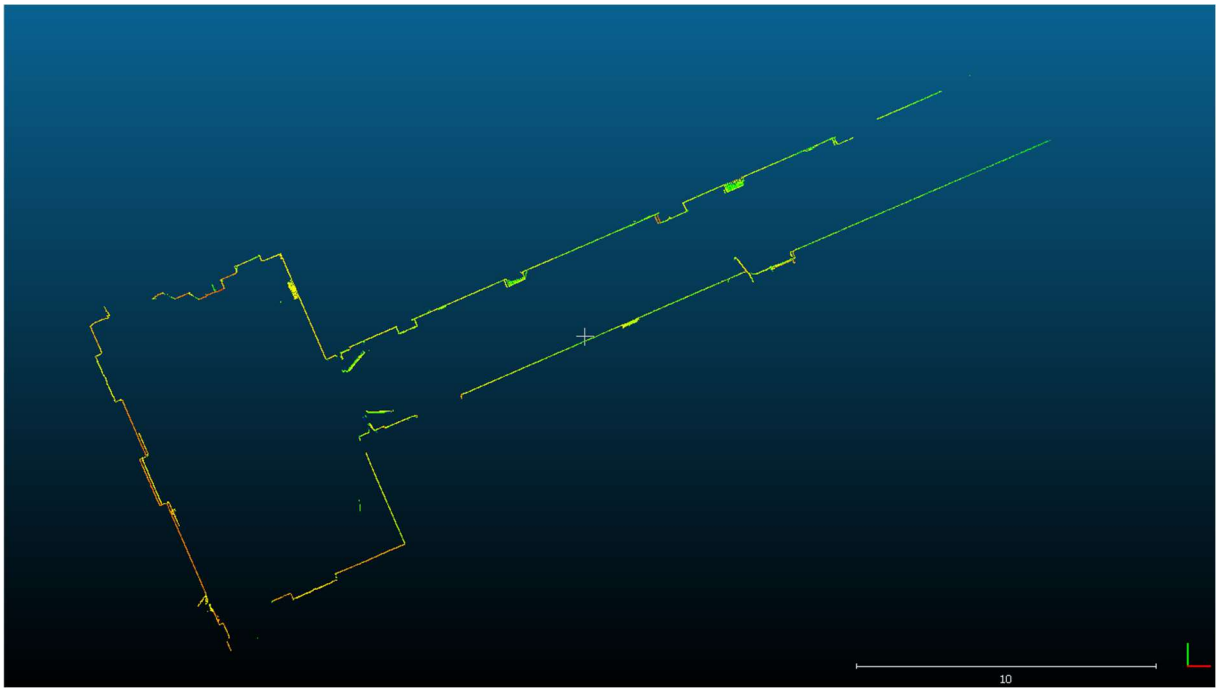
36. ábra A K épület FMT előtti folyosójáról előállt térkép

A külső folyosót egy jóval korábban készített pontfelhővel hasonlítottam össze.



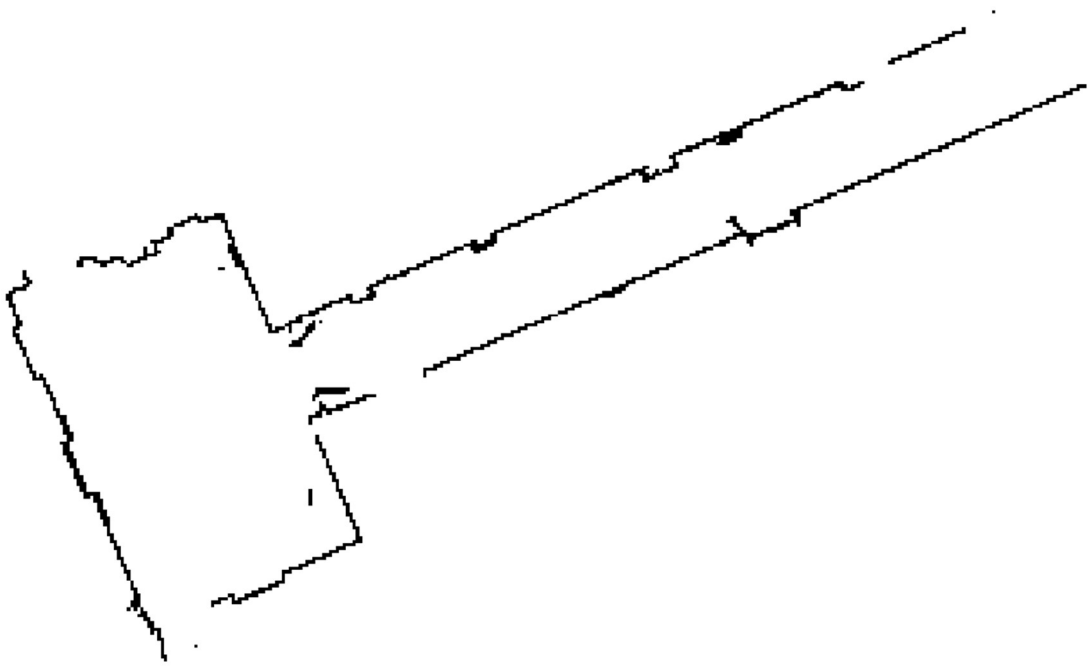
37. ábra A 3cm-re újramintavételezett pontfelhő-kivágat

Ebből egy metszetet készítettem a robot szkennerének megközelítő magasságában.



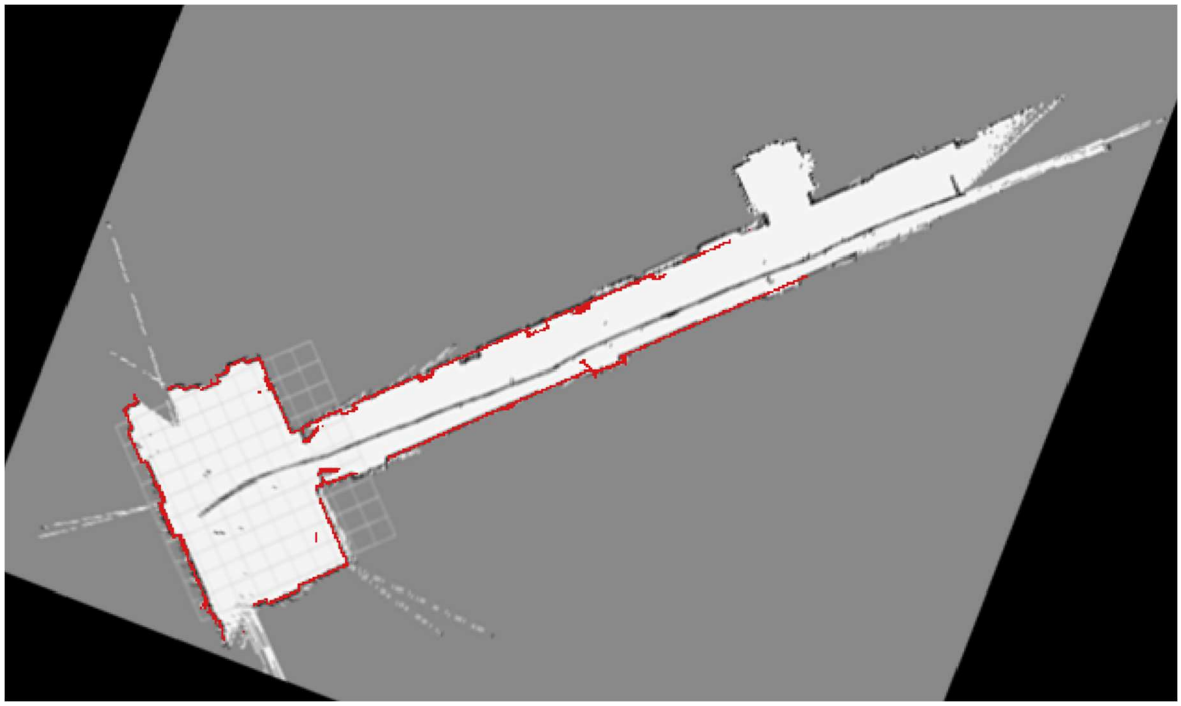
38. ábra A pontfelhő metszete

A pontfelhő metszetét ezután CloudCompare segítségével egy raszterré alakítottam, így már összehasonlítható lett a két raszteres formátum. A raszter 10 cm-es terepi felbontással készült.



39. ábra A pontfelhőből előállított raszter

A külső folyosóról előállított térképről és a pontfelhőből generált raszterről egy eltéréstérképet is készítettem. Ehhez először az előállított térképet metrikus rendszerbe georeferáltam a rácshálót felhasználva, majd be kellett illesztenem a pontfelhő koordináta rendszerébe. Ezt közös pontok alapján Helmert-transzformációval hajtottam végre.

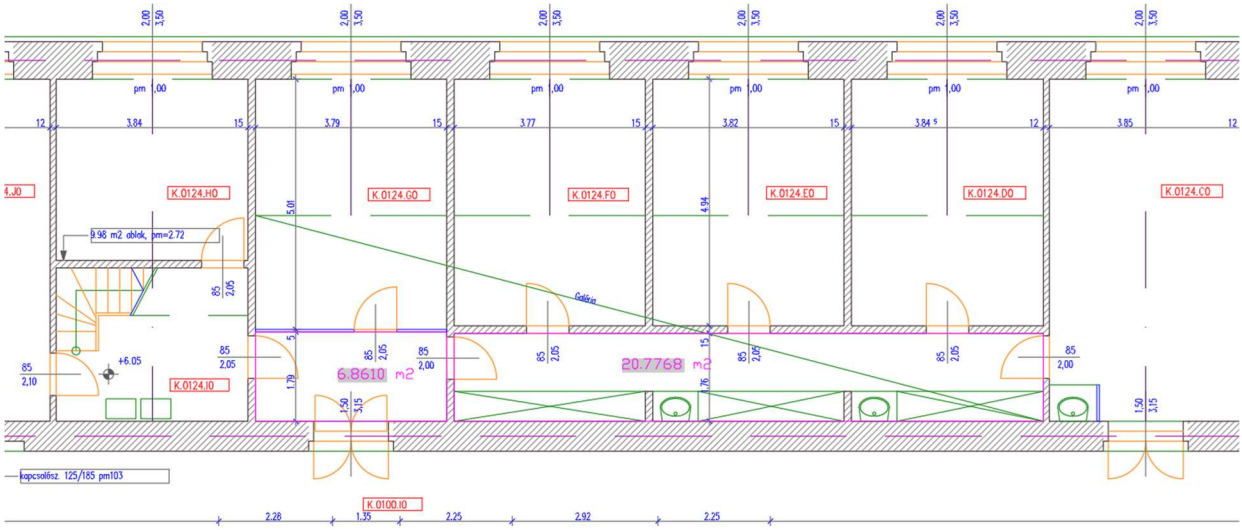


40. ábra A két raszter közös rendszerben, a pontfelhőből generált piros színnel

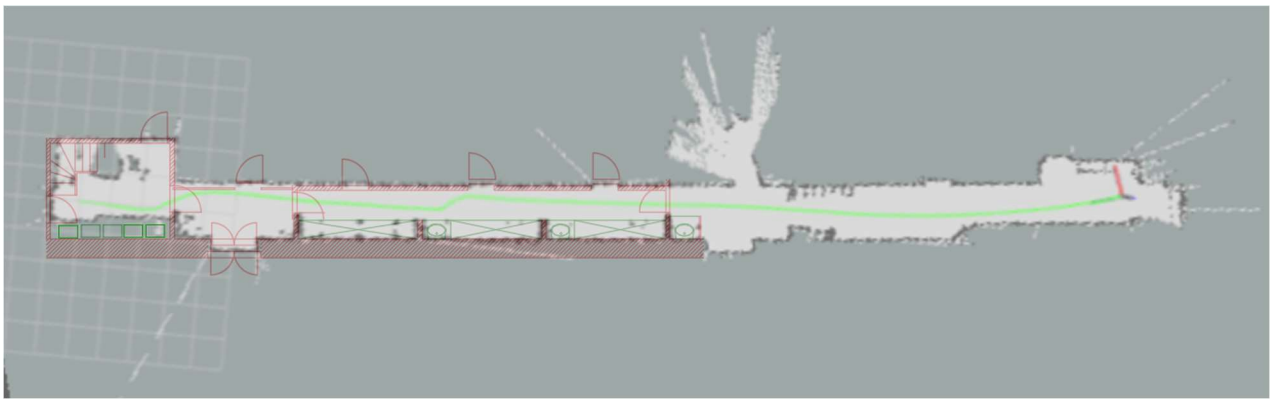
Az eltéréstérképet raszter kalkulátorral készítettem. Egyszerű raszter értékek egymásból való kivonásával. Az előállított térképet ehhez szürkeárnyalatossá alakítottam. A pontfelhőből előállított raszteren, ahol nincs pont, ott NaN értéket vesznek fel a pixelek, így ezen a helyen nincs értelme a kivonásnak. A többi helyen a két érték eltérésének nagyságából lehet az eltéréseket jelölni.

41. ábra Az előállított eltéréstérkép

A belső folyosóról előállított térképet egy korábbi felmérésből előállított alaprajzzal hasonlítottam össze.



42. ábra A terület alaprajzának részlete



43. ábra A közös koordinátarendszerbe rendezés után

Az alaprajzon rózsaszínnel jelölt két helység területét vizsgáltam.

Ezeknek területe az alaprajz szerint:

- 6.86 m^2
- 20.78 m^2

A georeferált térkép alapján QGIS-ben is számoltam területet; ezek négyzetméteren belül megegyeznek:

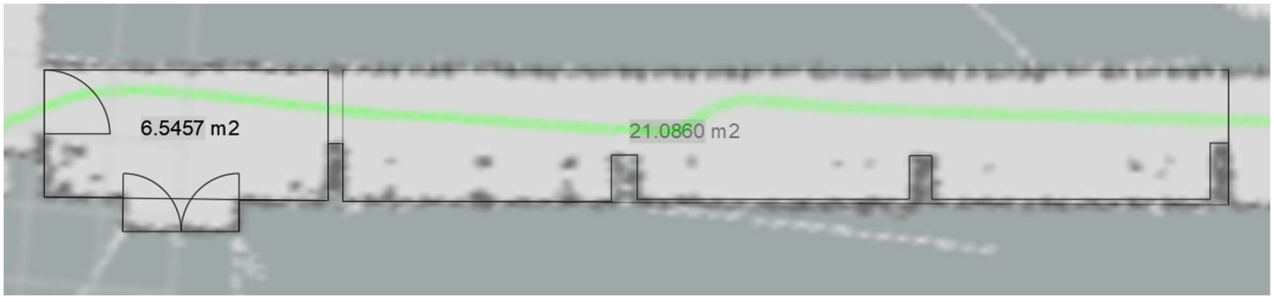
- 6.50 m^2
- 20.26 m^2

	id	terulet
1	1	6,494
2	2	20,265

44. ábra Terület QGIS-ben

AutoCAD-ben is megrajzoltam a szobák körvonalát, ott a következő eredményeket kaptam:

- 6.55 m^2
- 21.09 m^2



45. ábra AutoCAD rajz

8. Konklúzió, fejlesztési lehetőségek, gyengeségek

1.b Konklúzió

A platform működőképes, és wifi hálózat segítségével távolról is egyszerűen vezérelhető egy okostelefon segítségével. A SLAM algoritmus nagyon érzékeny a platform sebességére és a hirtelen fordulásokra. A hátsó szabad kerék még változtatásokat igényel. Térképek előállítása azonban már így is lehetséges, de a tereptől függően többszöri körbevezetés is szükséges.

Az előállított térképek jelenlegi formájukban a geodéziai pontosságot nem érik el, a hector_geotiff modullal, talán lehetséges lenne a 10 cm-es diszkretizációnál kisebb elérése is. Térinformatikai pontossági igényekre azonban meglehetősen pontos beltéri térképeket állíthatunk elő viszonylag gyorsan raszteres formában.

A rendszer néhány lehetséges alkalmazása lehet:

- olyan területekről viszonylag pontos térkép készítése, amik ember számára veszélyesek
- megvalósulási térkép előállítása bútorozatlan épületekről
- fa-kataszter előállítása (egyszerű körillesztéssel a fák pontos helyéhez, bár valószínűleg nem költséghatékony)
- irodákban berendezések helyének felmérése, például íróasztalok székek helyei
- régebbi épületek, amiknél már nem fellelhető az alaprajz, és nehezen mozdítható vagy mozdíthatatlan akadályok keletkeztek menekülési terv alaptérkép készítése

1.c Fejlesztési irányok

A fő fejlesztési irány, amit szeretnék vizsgálni, az a platform önjáróvá tétele. Léteznek a hector_mapping csomaghoz navigációs könyvtárak, illetve az ROS-nek is vannak beépített navigációs könyvtárai. Ennek vizsgálatára a TDK beadási határidejéig nem volt elegendő időm.

Érdemes lehet vizsgálni a platform segítségével más SLAM algoritmusokat is. Ehhez a platformot egyszerűen fel lehetne szerelni egy IMU-val, vagy kerekes odométerrel. Lehetséges lenne összehasonlítani az algoritmusok eredményeit, illetve a hector SLAM-et odometria alkalmazásával és nélküle.

1.d A rendszer hátulütői

A rendszer könnyedén becsapható a környezetből számított odometria miatt. Ha a platformot elindítanánk egy egyenes csőben, aminek az átmérője állandó, az algoritmus nem tudná megmondani, hogy halad-e előre, vagy egyhelyben áll, egészen míg nem észleli a cső végét. Ennek oka, hogy az illesztés és az odometria számítása alakjelző pontok alapján történik. Ahol, ilyenek nem találhatók az algoritmus nem tudja felismerni a platform mozgását.

A másik hátulütője, hogy közel vízszintes terepet igényel, ha nagyon meredek rámpára hajtanánk, a mögötte lévő talajt/padló egy nagyon közeli falnak értelmezné, ez fals térképhez vezet.

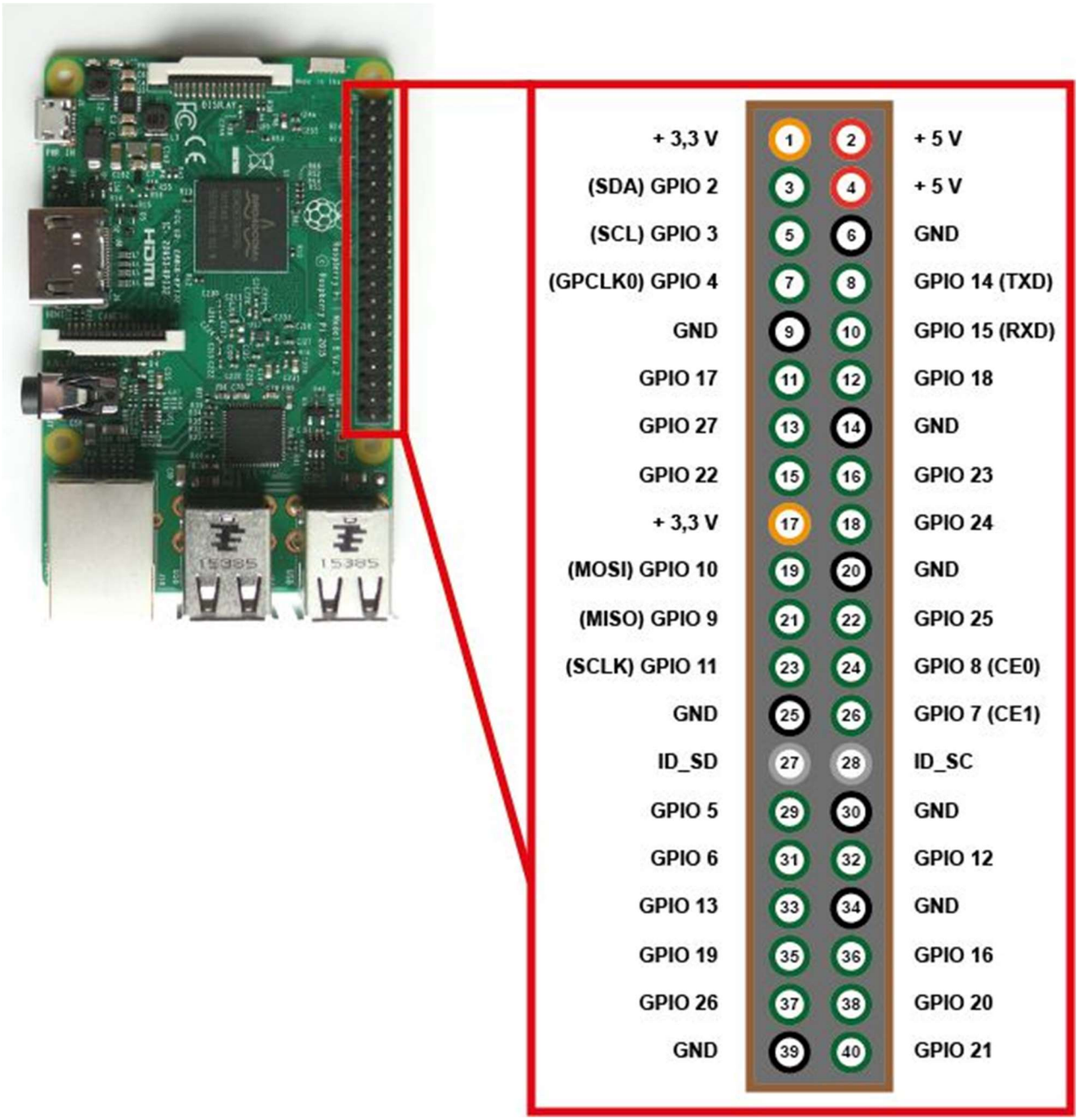
9. Köszönetnyilvánítás

Szeretném megköszönni a dolgozat elkészítéséhez nyújtott segítségét és fáradhatatlan munkáját konzulenseimnek, Dr. Barsi Árpádnak és Dr. Siki Zoltánnak. Továbbá szeretném megköszönni Sárdy Balázs villamosmérnöknek az áramkör számításainál és tervezésénél nyújtott segítségét.

10. Irodalomjegyzék

- Joseph Lentin (2015) – Mastering ROS for Robotics Programming
- https://en.wikipedia.org/wiki/Shakey_the_robot - *utolsó elérés: 2020.10.05* Shakey a robot
- Marcin Słomiany et al. (2020) – Motion Planning and Control of Social Mobile Robot – Part 1. Robot Hardware Architecture and Description of Navigation System: Progress in Automation, Robotics and Measurement Techniques
- Joao Machado Santos et al. (2013) – An Evaluation of 2D SLAM Techniques Available in Robot Operating System
- František Duchoň et al. (2019) – Verification of SLAM Methods Implemented in ROS
- http://wiki.ros.org/ *utolsó elérés: 2020.10.05* – ROS közösségi dokumentáció
- https://en.wikipedia.org/wiki/Robot_Operating_System – *utolsó elérés: 2020.10.05* Az ROS wikipédia szócikke
- Stuart Russell, Peter Norvig Panem Kft. (2005) – Mesterséges intelligencia
- https://blog.acolyer.org/2015/11/05/simultaneous-localization-and-mapping-part-i-history-of-the-slam-problem/ *utolsó elérés: 2020.10.05* – A SLAM története
- Kamarulzaman Kamarudin et al (2014) – Performance Analysis of the Microsoft Kinect Sensor for 2D Simultaneous Localization and Mapping (SLAM) Techniques https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4299068/ – *utolsó elérés: 2020.10.05*
- *https://hu.wikipedia.org/wiki/Raspberry_Pi – utolsó elérés: 2020.10.05 Raspberry wikipédia oldala*
- *https://static.raspberrypi.org/files/product-briefs/200521+Raspberry+Pi+4+Product+Brief.pdf – utolsó elérés: 2020.10.05 Raspberry kézikönyv*
- *https://github.com/makersdigest/T06-TB6612FNG-Motor-Controller/tree/master/raspberry-pi – utolsó elérés: 2020.10.05 motorvezérlő példakód, és motorvezerlő bekötés tippek*
- *https://www.slamtec.com/en/Lidar/A1Spec – utolsó elérés: 2020.10.05 motorvezérlő Az rplidar hivatalos oldala*

11. Mellékletek



46. ábra A Raspberry GPIO tűs panelje

Raspberry Pi 4 model B specifikációk:

Processzor:	Broadcom BCM2711, négy magos Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
Memória:	4GB LPDDR4
Csatlakozási lehetőségek:	2.4 GHz and 5.0 GHz IEEE 802.11b/g/n/ac wireless LAN, Bluetooth 5.0, BLE Gigabit Ethernet 2 × USB 3.0 ports 2 × USB 2.0 ports.
GPIO:	Szabványos 40-tűs GPIO ki- és bemenet
Videó és hang:	2 × micro HDMI ports (up to 4Kp60 supported) 2-lane MIPI DSI display port 2-lane MIPI CSI camera port 4-pole stereo audio and composite video port
Multimédia:	H.265 (4Kp60 decode); H.264 (1080p60 decode, 1080p30 encode); OpenGL ES, 3.0 graphics
SD kártya támogatás:	Micro SD foglalat az operációs rendszer háttértárának és adattárolásra
Tápellátás:	5V DC via USB-C csatlakozó (minimum 3A1) 5V DC via GPIO tűn keresztül (minimum 3A1) Power over Ethernet (PoE)– elérhető (külön tartozék szükséges hozzá)
Környezeti tűréshatár:	Működőképes 0–50ºC közt

A kameramodul adatlapja:

Specifikációk	**Camera Module v2**
Tömeg	3g
Felbontás	8 Megapixel
Videó felvételi módok	1080p30, 720p60 and 640 × 480p60/90
Linux integráció	V4L2 driver elérhető
Szenzor típusa	Sony IMX219
Szenzor felbontása	3280 × 2464 pixel
Szenzor felvételi terület	3.68 x 2.76 mm (4.6 mm átmérő)
Pixel méret	1.12 µm x 1.12 µm
Optika mérete	1/4"
Fókusztávolság	3.04 mm
Horizontális felvételi szög	62.2 fok
Vertikális felvételi szög	48.8 fok
Fókuszarány	2.0

Az motorvezérlő program python kódja:

```
from tkinter import *      # GUI library
from tkinter import font as tkFont # Large Arrow
from time import sleep     # Import sleep from time
import RPi.GPIO as GPIO    # Import Standard GPIO Module
GPIO.setmode(GPIO.BOARD)     # Set GPIO mode to BCM
GPIO.setwarnings(False);
# PWM Frequency
pwmFreq = 100
# Setup Pins for motor controller
GPIO.setup(12, GPIO.OUT)   # PWMA
GPIO.setup(18, GPIO.OUT)   # AIN2
GPIO.setup(16, GPIO.OUT)   # AIN1
GPIO.setup(22, GPIO.OUT)   # STBY
GPIO.setup(15, GPIO.OUT)   # BIN1
GPIO.setup(13, GPIO.OUT)   # BIN2
GPIO.setup(11, GPIO.OUT)   # PWMB
pwma = GPIO.PWM(12, pwmFreq)   # pin 18 to PWM
pwmb = GPIO.PWM(11, pwmFreq)   # pin 13 to PWM
pwma.start(100)
pwmb.start(100)
## Functions
################################################################################
def forward(spd):
   runMotor(0, spd, 0)
   runMotor(1, spd, 0)
def reverse(spd):
   runMotor(0, spd, 1)
   runMotor(1, spd, 1)



def turnLeft(spd):
```

```
    runMotor(0, spd, 0)
    runMotor(1, spd, 1)
def turnRight(spd):
    runMotor(0, spd, 1)
    runMotor(1, spd, 0)
def runMotor(motor, spd, direction):
    GPIO.output(22, GPIO.HIGH);
    in1 = GPIO.HIGH
    in2 = GPIO.LOW
    if(direction == 1):
        in1 = GPIO.LOW
        in2 = GPIO.HIGH
    if(motor == 0):
        GPIO.output(16, in1)
        GPIO.output(18, in2)
        pwma.ChangeDutyCycle(spd)
    elif(motor == 1):
        GPIO.output(15, in1)
        GPIO.output(13, in2)
        pwmb.ChangeDutyCycle(spd)
def motorStop():
    GPIO.output(22, GPIO.LOW)
root = Tk()
root.title("LandLose")
motorStop()
helv36 = tkFont.Font(family='Helvetica', size=36, weight=tkFont.BOLD)
```

```
forw = Button(root, text = "\u2191", font=helv36, padx = 50, pady = 50)
forw.grid(row=1, column=2, padx=10, pady=10)
backw = Button(root, text = "\u2193", font=helv36, padx = 50, pady = 50)
backw.grid(row=2, column=2, padx=10, pady=10)
left = Button(root, text = "\u2190", font=helv36, padx = 50, pady = 50)
left.grid(row=2, column=1, padx=10, pady=10)
right = Button(root, text = "\u2192", font=helv36, padx = 50, pady = 50)
right.grid(row=2, column=3, padx=10, pady=10)
forw.bind('<ButtonPress-1>',lambda event, a=55: forward(a))
forw.bind('<ButtonRelease-1>',lambda event: motorStop())
backw.bind('<ButtonPress-1>',lambda event, a=55: reverse(a))
backw.bind('<ButtonRelease-1>',lambda event: motorStop())
left.bind('<ButtonPress-1>',lambda event, a=45: turnLeft(a))
left.bind('<ButtonRelease-1>',lambda event: motorStop())
right.bind('<ButtonPress-1>',lambda event, a=45: turnRight(a))
right.bind('<ButtonRelease-1>',lambda event: motorStop())
root.mainloop()
```