ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

DOKTORI (PHD) ÉRTEKEZÉS

Somlyai László

Mobilrobot környezetének valósidejű térképezése

Témavezető: Dr. habil. Vámossy Zoltán

ALKALMAZOTT INFORMATIKAI ES
ALKALMAZOTT MATEMATIKAI
DOKTORI ISKOLA

Budapest, 2024. szeptember 18.

Komplex vizsga bizottság:

Elnök:
Prof.Dr.habil. Fullér Róbert
Tagok:
Prof. Dr. habil. Takács Márta
Prof. Dr. habil. Molnár András

Nyilvános védés teljes bizottsága:

Oponensek:
Dr. Pletl Szilveszter
Prof. Dr. habil. Szénási Sándor

Elnök:
Prof. Dr. Galántai Aurél DSc

Tagok:
Prof. Dr. habil. Takács Márta
Prof. Dr. habil. Fullér Róbert,
Prof. Em. Dr. Sima Dezső DSc

Titkár:
Dr. habil. Katona József

Nyilvános védés időpontja:
2024.

Tartalomjegyzék

Nevezéktan

Rövidítés	**Jelentés**
ATE	Absolute Trajectory Error
CAN	Controller Area Network
DIF	Depth Image Filter method
FAST	FAST feature detector
GPS	Global Positioning System
G^2o	Framework for graph-based optimization
HSV	Hue Saturation Value color representation
ICP	Iterative Closest Point
IMU	Inertial Movement Unit
ISVD	Iterative SVD method
LCD	Liquid-Crystal display
LC	Loop Closure
LIDAR	Light Detection and Ranging
LoG	Laplacian of Gaussian edge detector
ORB	Oriented FAST and rotated BRIEF
PC	Personal Computer
PCA	Principal Component Analysis
QR	Quick response code
RANSAC	RANdom SAmple Consensus
RC	Remote Control
RGB	Red-Green-Blue, color representation
RGB-D	RGB+Depth sensor
RMSE	Root mean square error
RPE	Relative pose error
SBA	Sparse Bundle Adjustment
SIFT	Scale-invariant feature transform
SLAM	Simultaneous Localization and Mapping
SURF	Speeded up robust features
SVD	Singular value decomposition
TOF	Time-of-Flight
TUM	Technical University of Munich
UAV	Unmanned aerial vehicle
VO	Visual Odometry

Rövidítés	**Mérték-egység**	**Jelentés**
P, Q	[-]	3D ponthalmazok
$f(x,y)$	[-]	kép, ahol x és y a képpont két koordinátája
$g(x,y)$	[-]	kép, ahol x és y a képpont két koordinátája
$h(x,y)$	[-]	kép, ahol x és y a képpont két koordinátája
w	[-]	képszélesség
h	[-]	képmagasság
$I_r(x,y)$	[-]	színes kép vörös csatornája
$I_g(x,y)$	[-]	színes kép zöld csatornája
$I_b(x,y)$	[-]	színes kép kék csatornája
k	[-]	küszöbérték, skalár
$[u_d, v_d]^T$	[-]	mélységi kamera pixelkoordinátája
$\delta(u_d, v_d)$	[mm]	a pixel mélységi értéke
$[u_c, v_c]^T$	[-]	színes kamera pixelkoordinátája
$[q_x, q_y, q_z, q_w]^T$	[-]	valós orientáció quaternionban megadva
$p_d = [x_d, y_d, z_d]^T$	[-]	3D pont a mélységi kamera koordináta-rendszerében
$p_c = [x_c, y_c, z_c]^T$	[-]	3D pont a színes kamera koordináta-rendszerében
$p_w = [x_w, y_w, z_w]^T$	[-]	3D pont a világ koordináta-rendszerében
$p_{wgt} = [x_{wgt}, y_{wgt}, z_{wgt}]^T$	[-]	3D pont a világ koordináta-rendszerében
$X_d = \{p_{d_1}, p_{d_2}, ..., p_{d_n}\}$	[-]	pontok halmaza a mélységi kamera rendszerében
$X_w = \{p_{w_1}, p_{w_2}, ..., p_{w_n}\}$	[-]	világ koordináta-rendszerben ábrázolt pontok
$X_{wgt} = \{p_{wgt_1}, p_{wgt_2}, ..., p_{wgt_n}\}$	[-]	világ koordináta-rendszerben ábrázolt pontok halmaza, valós pozíció
$f_c = [f_{cx}, f_{cy}]$	[-]	színes kamera fókusza
$p_c = [p_{cx}, p_{cy}]$	[-]	színes kamera főpontja
$f_d = [f_{dx}, f_{dy}]$	[-]	mélységi kamera fókusza
$p_d = [p_{dx}, p_{dy}]$	[-]	mélységi kamera főpontja
Twn	[-]	a jármű póza a kiindulási helytől nézve

1 Bevezetés

1.1. SLAM rendszerek

Napjainkban egyre elterjedtebbek az autonóm mobil robotok és az élet egyre több területén találkozhatunk velük. A feladatuktól függően lehet hogy ismerniük kell a környezetüket, észlelniük kell a felbukkanó akadályokat, meg kell tudniuk határozni a helyzetüket egy adott területen. Az egyes kutatások a robotok irányításához különféle típusú szenzorokat használnak fel, de a mobil robotok jelentős piaci térhódítása miatt már iparági szinten elterjedt szenzoros megoldásokról is beszélhetünk [1], [2].

A környezet érzékelésére az egyik elterjedt kutatási vonal a a gépi látás alapú technika. A jármű számára nem biztos, hogy rendelkezésre áll a környezetről térkép, így menet közben kell azt elkészíteni a bejárt területről szerzett információk alapján. Amikor a navigáció során a környezet megismerése és a saját helyzet meghatározása egyszerre történik, azt a navigációs eljárást nevezik Szimultán Lokalizáció és Térképmeghatározásnak, vagy a szakirodalomban alkalmazott rövidítéssel: SLAM rendszernek. A SLAM technika [3] célja, hogy a robot helyzetének és orientációjának folyamatos meghatározása mellett a környezetét is fel tudjuk térképezni.

A SLAM alapvető problémája az egyes mérésekből eredő bizonytalanságok kezelése, amelyek legfőképpen a szenzorok mérési pontatlanságának, a zajoknak és/vagy a technikai korlátoknak köszönhetők. Széles körben alkalmaznak különböző valószínűségi-modelleket ezeknek a bizonytalanságoknak a megfelelő közelítésére és a hatékony becslésekhez. A folyamat általánosságban véve olyan adatokkal operál, amelyek szenzoros (például LIDAR) mérésekből adódnak [4]. A kutatásom során alkalmazott robot esetén a választott szenzor egy RGB-D kamera, amely színinformáció mellett az egyes képpontok, képsíktól mért távolsági adatát is szolgáltatja. A kutatás alapját képző RGBD-alapú kamerának köszönhetően, nem csak egyszerűen SLAM-ről beszélhetünk, hanem Visual Simultaneous Localization and Mapping-ről (VSLAM-ról).

A beltéri navigáció során nem lehet használni olyan abszolút helyzetmeghatározásra szolgáló rendszereket, mint például a GPS rendszer, ami nagymértékben megkönnyíti a nagyobb területen való helyzetmeghatározást. Azonban kiépíthető beltérben is használható

abszolút helyzetmeghatározó rendszer, de ehhez külső szenzorok, vagy passzív referencia jelölők elhelyezése szükséges. A kutatásom elsősorban a beltéri helyzetmeghatározásra használható módszereket vizsgálja és mutat be egy saját, beltérben használható SLAM rendszert, aminek nincs szüksége külső szenzorokra, jeladókra. Az elmúlt időben egyre több gazdaságos és kis méretű szenzor érhető el gépi látást használó rendszerekhez, például a Xtion [5], Intel RealSense [6] és Kinect [7] szenzor. Az egyszerűbb kivitelük ellenére elegendő pontossággal rendelkeznek egy kisebb méretű robot környezetének érzékeléséhez. A jármű környezetéről egy részletgazdag 3D térkép készíthető segítségükkel. Az RGB-D kamerákat, akárcsak a sztereókamerákat vagy a LIDAR-t igen sok kutatás használja, mint elsődleges szenzort.

Ilyen SLAM rendszerket dolgoztak ki Scaramuzza-ék [8], Whelan és kollégái [9] [10], Qiang csapata [11], Endres-ék [12] [13], Stücklerék [14], Henryék csapata [15] [16], Hachiuma-ék [17], Yangdong-ék [18] és Rongsong-ék [19]. Valamint ilyen rendszer a Co-fusion [20] és a Static Fusion [21]. A felsorolt munkákat vettem a kutatásom alapjául és a saját rendszeremet elsősorban ezekkel a munkákkal hasonlítottam össze minőségi jellemzőkben.

1.2. Saját rendszer fejlődése

A disszertációban bemutatott saját SLAM rendszer és 3D rekonstrukciós eljárás esetében nincs szükség előre kiépített szenzorrendszerre. A rendszer a mozgás becslését egy RGB-D kamera adatainak felhasználásával éri el. A jelenleg kifejlesztett rendszernek számos előzmény projektje volt. Az első ilyen projekt 2008-ban a Magyarok a Marson című versenyen II. helyezést ért el. Itt egy plafonra szerelt kamera figyelte a járművet és ennek alapján történt az irányítása. Ennek továbbfejlesztett változata a 2009-es Országos Tudományos Diákköri Konferencia, Műszaki tudományok szekciójában első helyezésben részesült (”Mavridisz Vaszilisz, Somlyai László, Gál Béla: Lézerszkennerrel támogatott körbelátórendszer önjáró roboton”). Itt egy saját, 360 fokban körbelátó lézerszkennert fejlesztettünk ki a navigációhoz. Ezekben a projektekben fő szerepem a hardvert működtető elektronika megtervezése és az azt működtető firmware elkészítése volt. A robotos rendszerünket lényegesen továbbfejlesztve mutattuk be a 2011-es OTDK Informatkatudományi szekcióban, ahol dolgozatunkat (Csaba György, Somlyai László: Gépi látáson alapuló akadályelkerülés) II. helyezéssel értékelték. Itt a korábbi lézerszkennert fejlesztettük tovább, ami egy LEGO tornyon kapott helyet.

A rendszerünket egy távirányítós autóra építettük fel. Az autó eredeti vezérlő rendszere helyett két elektronikai kártyát terveztem, aminek a segítségével az autó számítógépről

irányítható volt. Ezek a CAN bridge nevű kártya (6.1.3. melléklet) és a motormeghajtó modul (6.1.3. melléklet). A rendszerünket két magyar nyelvű előadáson [22] [23] és egy nemzetközi konferencián [24] is bemutattuk.

Kutatásaim során dolgoztam egy kisméretű GPS által vezérelt robotrepülő kommunikációján [25] és készítettem egy antennaforgató rendszert kisméretű robotrepülőkhöz, ahol a GPS rendszerrel megismerkedtem. Később készítettem egy légköri paramétereket mérő adatgyűjtő rendszert [26], amit kis méretű robotrepülőn, vagy robotautón lehettett elhelyezni. Egy későbbi munka során összehasonlítottuk a saját strukturált megvilágítást használó érzékelőt, amit korábban fejlesztettünk ki a Kinect szenzorral, valamint ezek pontosságát is megvizsgáltuk [27].

A korábban kifejlesztett lézerszkennert felváltottam a Kinect szenzorral, a nagyobb mérési pontosság és nagyobb mérési adatmennyiség miatt. Ezt a szenzort kalibráltam, majd elhelyeztem a saját robotautóra [28]. Itt vizsgáltam a szenzor pontosságát, és javitottam a mélységi képet szűrés segítségével. Az összeépített rendszer segítségével az Egyetem laborjaiban készítettem számos off-line adathalmazt a Kinect szenzorral, amit később az algoritmusaim teszteléséhez tudtam felhasználni.

A mobilrobot és rá rögzített szenzor mozgásának becslésére az első probálkozásokat egy SVD felbontáson alapuló iterativ algoritmussal végeztem. Az egymás utáni színes képeken kiválasztott jellemzőpontpárok segítségével, az SVD felbontást használva határoztam meg két egymás utáni kamera pozició közötti relatív elmozdulást [29]. Az eljárás, még nagy futási idővel rendelkezett.

Ennek az algoritmusnak a továbbfejlesztésével jelentős javulást értem el a futási idő és pontosság tekintetében [30] [31]. Végül kifejlesztettem egy saját keretrendszert a jelenlegi ISVD algoritmusomhoz [32]. A keretrendszer segítségével a saját adathalmazokon kívül, két külső adatbázison (TUM [33], POZNAN [34]) is lehet az algoritmus pontosságát vizsgálni.

A rendszer következő változata pontosabb becslést ad az elmozdulásra, és a működése is robusztusabb a korábbi rendszernél, a továbbfejlesztett illesztési eljárásnak és a valós idejű Loop-closure (zárt hurok detektálási) algoritmusnak köszönhetően [35]. Végül a végső rendszerem pontosságát a több korábbi kép súlyozott illesztésével, az ISLAM algoritmusom segítségével tovább pontosítottam [36].

2 Robot helymeghatározási rendszerek, SLAM eljárások

2.1. A kutatási téma háttere

A térképkészítés és a lokalizáció meghatározása érdekében a mobilrobotra szerelt, vagy általánosabban, a mozgó merevtesten elhelyezkedő kamera egymás utáni felvételeket készít, miközben elmozdul a térben. A két ponthalmazt, vagy szokásos elnevezéssel pontfelhőt egymásba térképező tarnszformáció meghatározza a két felvétel között történt elmozdulást és orientációváltozást. Pontfelhők illesztésére az egyik tradicionális és gyakran alkalmazott technika az Iterative Closest Points (ICP) algoritmus [37], melynek fő lépései a következők: Adott két ponthalmazunk P és P', és a P' ponthalmazhoz szeretnénk illeszteni az előző képből kinyert P ponthalmazt. A két ponthalmaz súlypontját meghatározzuk, és egy eltolás transzformációval az újabb ponthalmaz súlypontjába toljuk a korábbit. P minden pontjához megkeressük P' legközelebbi pontját, illetve P' minden pontjával is megcsináljuk ugyanezt. Ahol mindkét ponthalmazból vizsgálva ugyanaz a pontpárt kapjuk, meghatározzuk azt a transzformációt, ami a legkisebb hibával képezi P-t P'-be, majd iteráljuk a pontpárok megfeleltetését és az egymásba képzést. Az algoritmusnak számos változata van [38] és RGB-D szenzorok esetén is sokszor alkalmazták már [39, 40].

Ezeket a változatokat a következő hat szakasz egyikére gyakorolt hatásuk alapján osztályozhatjuk a [41] cikk alapján:

1. Egy pontkészlet kiválasztása az egyik vagy mindkét pontfelhőben.
2. Ezen pontok párosítása a másik háló mintáival.
3. A pontpárok illesztésének súlyozása, az illesztés minőségének függvényében.
4. Bizonyos pontpárok elvetése, akár egyenkénti vizsgálat, akár az összes pár figyelembevétele alapján.
5. Hibametrika hozzárendelése a pontpárok illesztési minősége alapján.

6. A hibametrika minimalizálása.

A pontkészlet kiválasztásához több stratégia is létezik: az összes elérhető pont használata, amely nagyon számításigényes; az elérhető pontok egyenletes mintavételezése; véletlen mintavételezés (minden iterációban más mintahalmazzal); pontok kiválasztása magas intenzitásgradiens alapján, olyan változatokban, amelyek a szín vagy intenzitás használatával segítik az illesztést; az előző módszerek mindegyike választhat pontokat csak az egyik pontfelhőből, vagy kiválaszthat forráspontokat mindkét halmazból. Számos kutatás un. jellemzőpont, vagy jellemződetektor alapú kiválasztást végez, ahol valamilyen lokális specialitás alapján történik a pontok kiválasztása. Az eredmény lehet egyszerűen a jellemzőpontok koordinátája, de jellemződetektorok esetén leíróvektort is eltárolhatnak a pozíció mellett, ami a későbbi illesztést segítheti. Az utóbbiak gyors működése és hatékonysága miatt a saját rendszerben ilyen eszközöket teszteltem és alkalmaztam.

A kutatások irányát meghatározza, hogy milyen jellemzőpont detektort használnak, mivel azok a különböző kép-transzformációkra más és más módon reagálnak. Az egyszerűbb jellemződetektorok (FAST [42]), futási ideje jóval kisebb, mint a bonyolultabb detektoroké (SIFT [43], PCASIFT és SURF [44]), de az egyes térbeli transzformációkra kevésbé invariánsak [45]. A SIFT detektor adja az egyik legjobb eredményt több transzformációra és az esetleges képi elmosódásokra [46]. A SIFT hátránya a mobil robotok kérdéskörében a magas feldolgozási idő. A valósidejű térképépítéshez elengedhetetlen feltétel a gyors jellemzőpont kereső algoritmus kiválasztása. Több cikkben is foglalkoznak az egyes detektorok tesztelésével, én is összehasonlítottam több detektort is, így például az ORB [47], SIFT, SURF és FAST módszereket [31]. A robusztus működés és a gyorsaság alapján választottam közülük a vizsgálatok során.

A jellemzőpontok meghatározása mellett a megoldás kulcslépése, hogy az egymás után készített képeken meghatározott jellemzőpontokat, azokak térbeli pozicióját felhasználva, miként párosítjuk, azaz milyen transzformáció segítségével képezzük egymásba őket.

2.2. Pontfelhők illesztése SVD eljárás segítségével

Az irodalom szerint az Iteratve Closest Points (ICP) algoritmus egy hatékony megoldása Arun, Huang és Blostein publikációján [48] alapszik. A fejezetben a cikk alapján foglalom össze a módszer lényegét.

Számos számítógépes látási alkalmazásban, nevezetesen egy merev test mozgási paramétereinek becslésénél 3D pontok megfeleltetései alapján [49], valamint egy merev test egy referenciahoz képesti relatív helyzetének meghatározásánál [50], a következő

matematikai problémával találkozunk. Adott két ponthalmaz $P = \{p_i\}$ és $P' = \{p'_i\}$, $i = 1, 2, ..., N$, ahol p_i és p'_i 3×1 méretű oszlopvektorok. A két ponthalmaz P és P' elemei között a következő kifejezés teremt kapcsolatot:

$$p'_i = Rp_i + T + N_i \tag{2.1}$$

ahol R 3×3-as forgatómátrix, T az eltolásvektor (3×1-es oszlopvektor) és N_i egy zajvektor. Feltételezzük a modellben, hogy a forgatás origón áthaladó tengely körül történik. Keressük R és T értékét, mely minimalizálja a következő kifejezést

$$E = \sum_{i=1}^{N} \|p'_i - (Rp_i + T)\|^2 \tag{2.2}$$

A megoldás meghatározására iteratív módszert adott Huang, Blostein és Margerum [51]. Faugeras és Herbert [52] nem iteratív megoldása kvaterniókat használt. Arunék megoldása sem iteratív, amely 3×3-as mátrixok szinguláris érték dekompozícióját (SVD) alkalmazza. A megoldás előnye, hogy a számítógépes időigénye más módszerekhez képest általában kisebb.

2.2.1. Az eltolás és a forgatás szétválasztása, majd a forgatómátrix számítása SVD módszerrel

Arunék felhasználták kutatócsoportjuk korábbi, [51]-ban publikált eredményét: Ha (2.2) legkisebb négyzetek módszer szerinti megoldása $\hat{R}$ és $\hat{T}$, akkor a $P = \{p'_i\}$ és $P'' = \{p''_i\} = \{\hat{R}p_i = \hat{T}\}$ ponthalmazoknak ugyanaz a súlypontja, például

$$p' = p'', \tag{2.3}$$

ahol

$$p' = \frac{1}{N} \sum_{i=1}^{N} p'_i \tag{2.4}$$

$$p'' = \frac{1}{N} \sum_{i=1}^{N} p''_i = \hat{R}p + \hat{T} \tag{2.5}$$

$$p = \frac{1}{N} \sum_{i=1}^{N} p_i \tag{2.6}$$

Áttérve súlyponti koordinátákra,

$$q_i = p_i - p \tag{2.7}$$

$$q'_i = p'_i - p' \tag{2.8}$$

és a (2.2) kifejezésbe behelyettesítve, kapjuk

$$E = \sum_{i=1}^{N} \|p'_i - (Rp_i + T)\|^2 = \sum_{i=1}^{N} \|q'_i + p_i - (R(q_i + p) + T)\|^2 = \sum_{i=1}^{N} \|q'_i - Rq_i\|^2 \tag{2.9}$$

Ezért az eredeti legkisebb négyzetek szerinti probléma két részre redukálódik:

1. Az $\hat{R}$ forgatómátrix meghatározása, amely minimalizálja E értékét a (2.9) kifejezésben.

2. Majd az elmozdulás értékének már nyílvánvaló kiszámítása:

$$\hat{T} = p' - \hat{R}p \tag{2.10}$$

A 1. részfeladat megoldására Arunék által leírt módszert az 1. algoritmus mutatja be. Mindkét ponthalmaz esetében súlyponti koordinátarendszerre térünk át. Az így kapott értékekeből egy kovariancia mátrixot konstruálunk, amelynek a következő lépésben elkészítjük az SVD felbontását $H = U\Lambda V^T$, majd kiszámoljuk $X = VU^T$ mátrixot. Ha X sajátértéke, $det(x) = +1$, akkor X a keresett forgatómátrix. Amennyiben a determináns értéke -1, akkor elfajuló esetet kell kezelni. Ez ritkán, akkor fordulhat csak elő [48] szerint, ha az adatok zajosak ($N_i \neq 0$).

Umeyama [53] cikkében továbbfejlesztette Arun et. al. fenti SVD megoldását. Egyrészt a transzformációt kibővítette skalárral való skálázással, azaz euklideszi transzformáció helyett általánosabb, hasonlósági transzformációt tárgyal, másrészt a pontvektorok dimenzióját sem kötötte meg 3D-re. Legfontosabb eredménye pedig, hogy mindig megadja zárt alakban a helyes transzformációs paramétereket, még akkor is, ha az adatok sérültek, zajjal terheltek.

Legyenek A és B $m \times n$ méretű mátrixok és R $m \times m$ forgatómátrix. Az AB^T SVD felbontása legyen UDV^T, ahol ($UU^T = VV^T = I, D = \mathrm{diag}(d_i), d_1 \geq d_2 \geq \cdots \geq d_m \geq 0$). Ekkor az $\|A - (RB)\|^2$ R forgatásra vonatkozó minimum értéke:

$$\min \|A - (RB)\|^2 = \|A\|^2 + \|B\|^2 - 2\mathrm{tr}(DS) \tag{2.11}$$

ahol tr(DS) a mátrix főátlójában elhelyezkedő elemek összege és

$$S = \begin{cases} I & \text{ha } \det(AB^T) \geq 0 \\ \text{diag}(1, 1, \cdots, 1, -1) & \text{ha } \det(AB^T) < 0 \end{cases} \tag{2.12}$$

Ha $\text{rank}(AB^T) \geq m - 1$, akkor az optimális R forgatómátrix, amely a minimális értéket szolgáltatja:

$$R = USV^T \tag{2.13}$$

ahol S értékét a (2.13) kifejezésben a következő módon kell megválasztani:

$$S = \begin{cases} I & \text{ha } \det(U)\det(V)\text{=1} \\ \text{diag}(1, 1, \cdots, 1, -1) & \text{ha } \det(U)\det(V)\text{=-1} \end{cases} \tag{2.14}$$

és ekkor $\det(AB^T) = 0$, valamint $\text{rank}(AB^T) = m - 1$.

2.2.2. A forgatási mátrix átalakítása kvaternióvá és Euler szögekké

Ha a mozgáskövető rendszer a kamera orientációját rotációs mátrixként R adja meg, ezt át lehet alakítani egységkvaternióvá. A kvaternió $q = [q_w, q_x, q_y, q_z]^T$ a forgást reprezentálja, és normalizált kvaterniót (egységkvaterniót) használunk. Az átalakítás a rotációs mátrixból R kvaternióvá a (2.19) – (2.23) képletek szerint történik.

Legyen a rotációs mátrix:

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{2.19}$$

Ekkor a kvaternió komponensei:

1. q_w komponens:

$$q_w = \frac{1}{2}\sqrt{1 + r_{11} + r_{22} + r_{33}} \tag{2.20}$$

2. q_x komponens:

$$q_x = \frac{r_{32} - r_{23}}{4q_w} \tag{2.21}$$

3. q_y komponens:

$$q_y = \frac{r_{13} - r_{31}}{4q_w} \tag{2.22}$$

1. algoritmus SVD algoritmus $\hat{R}$ és $\hat{T}$ meghatározására – Arun et. al. alapműve alapján.

Input: $\{p'_i\}$, $\{p_i\}$ az aktuális és az azt megelőző pontfelhő
Output: $\hat{R}$ az illesztésből becsült elfordulás mátrix
Output: $\hat{T}$ az illesztésből becsült elmozdulás vektor

1: p, p' számtása (2.6) és (2.4) szerint
2: $\{q_i\}$, $\{q'_i\}$ súlyponti koordinátákra áttérés (2.7) és (2.8) szerint
3: Határozzuk meg a következő 3×3 kovarianciamátrixot

$$H = \sum_{i=1}^{N} q_i q_i'^T \tag{2.15}$$

ahol ^{T} felsőindex a transzponálást jelenti.

4: H mátrix SVD felbontása

$$H = U \Lambda V^T \tag{2.16}$$

5: X számítása

$$X = V U^T \tag{2.17}$$

6: X determinásértékének meghatározása: $det(x)$
7: **if** $det(x) = +1$ **then** $\hat{R} = X$
8: **if** $det(x) = -1$ **then** X egy tükrözés.
9: a) Ha a H mátrix egyik szinguláris értéke (például λ_3) nulla. Ekkor a kívánt forgatás a következőképpen található meg: $X' = V' U^T$, ahol V' a V mátrixból származik, azzal a különbséggel, hogy a 3. oszlop előjelét megváltoztatjuk.
10: b) Ha a H mátrix egyik szinguláris értéke sem nulla, akkor a hagyományos legkisebb négyzetek módszere valószínűleg nem megfelelő. Ebben az esetben RANSAC-szerű technikához kel fordulni. ▷ Umeyama bizonyítása [53] azonban mindig megadja zárt alakban a helyes transzformációs paramétereket, még akkor is, ha az adatok sérültek, vagy zajjal terheltek.
11: $\hat{T}$ elmozdulás meghatározása

$$\hat{T} = p' - \hat{R} p \tag{2.18}$$

4. q_z komponens:

$$q_z = \frac{r_{21} - r_{12}}{4q_w} \tag{2.23}$$

Amennyiben a forgatást Euler-szögek segítségével szeretnénk leírni, mint később az eredmények bemutatásánál én is ezt használom, és például Roll (ϕ) az X-tengely körüli, Pitch (θ) az Y-tengely körüli és Yaw (ψ) a Z-tengely körüli forgatás, ekkor az R forgatómátrix:

$$R = R_z(\psi)R_y(\theta)R_x(\phi) \tag{2.24}$$

ahol:

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \tag{2.25}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \tag{2.26}$$

$$R_z(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.27}$$

A teljes forgatómátrix így:

$$R = \begin{bmatrix} \cos\psi\cos\theta & \cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi & \cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi \\ \sin\psi\cos\theta & \sin\psi\sin\theta\sin\phi + \cos\psi\cos\phi & \sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix} \tag{2.28}$$

Ha ismét adott az R forgatómátrix (2.19) szerint, a Roll (ϕ), Pitch (θ) és Yaw (ψ) szögek kiszámítása a következő:

1. Pitch (θ):

$$\theta = \arcsin(-r_{31}) \tag{2.29}$$

2. Yaw (ψ) (ha $\theta \neq \pm\frac{\pi}{2}$):

$$\psi = \arctan2\left(\frac{r_{21}}{\cos\theta}, \frac{r_{11}}{\cos\theta}\right) \tag{2.30}$$

3. Roll (ϕ) (ha $\theta \neq \pm\frac{\pi}{2}$):

$$\phi = \arctan2\left(\frac{r_{32}}{\cos\theta}, \frac{r_{33}}{\cos\theta}\right) \tag{2.31}$$

A különleges esetek kezelése fontos, amikor θ értéke $\pm\frac{\pi}{2}$, azaz ($\pm$90 fok) körül van, mivel ilyenkor az egyenletek szingularitásokhoz vezethetnek.

- Ha $\theta = \frac{\pi}{2}$:

$$\psi = 0, \quad \phi = \arctan2(r_{12}, r_{22}) \tag{2.32}$$

- Ha $\theta = -\frac{\pi}{2}$:

$$\psi = 0, \quad \phi = \arctan2(-r_{12}, -r_{22}) \tag{2.33}$$

A számításoknál javasolt a kétparaméteres arctan2 függvény használata, mert az síknegyednek megfelelő megoldást szolgáltat a szögre.

2.3. Homográfia transzformáció homogén koordinátákkal

A homográfia transzformáció egy projektív transzformáció, amely kétdimenziós síkok között hoz létre kapcsolatot. A homográfia egy mátrix formájában jelenik meg, és képes leírni, hogyan vetül át egy sík másik síkra, figyelembe véve perspektivikus torzításokat is. Ezt a transzformációt gyakran használják számítógépes látásban két kép közötti geometriai kapcsolat modellezésére, például két nézet közötti perspektíva illesztésére. A homográfia transzformáció számos területen alkalmazható:

- Képek regisztrációja: Különböző nézetekből készült képek geometriai transzformációval történő illesztése.
- Kamerák közötti vetületek transzformálása: Egy sík egyik kamerából a másikba vetíthető át homográfia segítségével.
- Panorámaképek készítése: Két vagy több kép összeillesztése közös jellemzőpontok alapján, a homográfia transzformáció segítségével.

Ezt a transzformációt egy 3×3-as mátrix írja le, amely perspektivikus vetítést végez a két sík között. A homográfia mátrix H általános alakja:

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \tag{2.34}$$

Ez a mátrix 8 független paramétert tartalmaz (a 9. elem egy arányossági tényező). A homográfia számításához legalább négy megfelelő pontpárra van szükség, hogy a mátrixot meghatározhassuk. A homográfia transzformáció leírására célszerű beevezetni a homogén koordináták használatát, ahol a pontokat eggyel magasabb dimanziós térben ábrázoljuk, azaz a 2D pontot 3D vektorként, a 3D pontot 4D vektorként. Az utolsó komponens a méretarány, vagy skálázó tényező. Tehát egy kétdimenziós pontot (x, y) homogén koordináták formájában eggyel magasabb dimenzióban a w úgynevezett skálázó tényező bevezetésével így írhatunk fel:

$$p = \begin{bmatrix} x \\ y \\ w \end{bmatrix} \tag{2.35}$$

A homográfia transzformáció egy p pontot egy másik pontba p'-be a következőképpen képezi át:

$$p' = Hp \tag{2.36}$$

Ez explicit módon felírva:

$$p' = \begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2.37}$$

A valós koordinátákhoz való visszatéréshez osztanunk kell a skálázó tényezővel, a harmadik koordinátával (w):

$$x' = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}}, \quad y' = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}} \tag{2.38}$$

A (2.38) képlet mutatja, hogyan történik a transzformáció a homogén koordináták segítségével.

2.4. Hasonló kutatások bemutatása

Az önálló SLAM rendszerek, ahol nincs szükség külső érzékelők telepítésére, különböző érzékelő fajtákat használnak fel. Az egyszerűbb kétdimenziós LIDAR-t használó rendszerek kisebb erőforrásigénnyel rendelkeznek, ennek köszönhetően nagy sebességgel képesek feldolgozni a szenzor adatokat. A 2D SLAM algoritmusok általában ICP, vagy point-to-plane eljárást használnak az elmozdulás becslésére.

Nagyobb erőforrásigényű háromdimenziós LIDAR, vagy gépi látást használó rendszerekkel azonban nagyobb pontosság érhető el, valamint részletgazdagabb térkép építhető fel. Azonban ezek a rendszerek nagyobb számítási kapacitást igényelnek és kisebb működési frekvenciára képesek.

Az egyik, teljesen gépi látáson alapuló SLAM rendszert Henry-ék csapata fejlesztette ki [15], [16]. A rendszer RGB-D kamera adatokon és jellemzőpont keresésen alapul. Az elmozdulás becsléséhez és az út gráf elkészítéséhez (pose graph) ICP alapú algoritmust használnak. Lecsökkentették az ICP algoritmus nagy számítási igényét, hogy ne legyen szükség a teljes pontfelhők illesztésére. Néhány előre jól kiválasztott pontot, redukált pontfelhőt szükséges illeszteni egymáshoz. A pose graph optimization [54] helyett, SBA eljárást [55] használtak a becsült útvonal hibáinak minimalizálására.

Egy másik rendszer hasonló megoldást használ [56], ahol egy robotrepülő irányítása RGB-D kamera alapján történik. FAST jellemzőpont detektort alkalmaznak, ami kevésbé robusztus detektor. Az algoritmus rövid feldolgozási idővel rendelkezik, de kevésbé invariáns a transzformációkra. A pontfelhők illesztéséhez a kezdeti elmozdulás becslésére egy IMU szenzor adatait használja fel. Majd több kutatás keretében a jellemzőpontok követésével javítottak az algoritmus pontosságán. A robosztus jellemződetektorok használatával tovább javítható a rendszer.

Az egyes jellemzőpont detektorok működése különböző környezetekben eltérő lehet. Egy kutatás három robusztus jellemzőkereső eljárást hasonlít össze [12], a SIFT, SURF és ORB jellemződetektort. A jellemzőpontokat mind az egymásutáni képek illesztéséhez, mind a G^2o optimalizációhoz [54] felhasználják. A térkép megjelenítésére az octree-based volumetric representation eljárást használják fel.

Egy másik kutatás [14] a surfel map-ot elmenti a pozíció becsléshez, illetve egy nagyobb felbontású surfel map-ot is tárol octree-ben a későbbi 3D felület rekonstruáláshoz. A G^2o optimalizálás után készül el a végső globális felületi térkép.

Bo-ék csapata egy ICP alapú eljárást mutat be a háromdimenziós térkép rekonstruálására [57]. A módszer először SURF jellemzőpontokat keres két színes képen, majd két lépésben megbecsüli a köztük lévő relatív elmozdulást. Első lépésként a jellemzők

euklideszi távolsága alapján törlik a nem megfelelő jellemzőpont-párokat. Következő lépésként RANSAC algoritmussal segítségével egy kezdeti elmozdulás becslést határoznak meg az egymást követő mérések között és törlik azokat jellemzőpárokat, amik hibásan lettek detektálva. A következő lépésben módosított ICP algoritmus segítségével határozza meg a végső becsült elmozdulást. A jellemzők véletlenszerű kiválasztását, úgy oldották meg, hogy a köztük lévő euklideszi távolság meghatározásával a túl közelieket elhagyták. A Loop-closure algoritmus működésékez kulcsképeket választanak ki. A kulcsképek kiválasztásának feltétele, hogy ne legyenek túl közel egymáshoz.

Eredményeiket összehasonlították Felix-ék SLAM rendszerének eredményeivel, akik [12] SIFT, SURF, ORB jellemzőpont detektorokat használtak fel. A jellemzőpont kereső eljárásokat sok kutatás vizsgálja, de gyakran a nagyobb számításigényű mellett döntenek azok robusztussága miatt. Például az ORB futása ugyan gyorsabb a SURF detektorénál, azonban hasonló munkákban mégis többször kerül kiválasztásra a SURF detektor [11], mert alig csökken a pontossága, ha ezt használják, de robosztusabb a működése.

A gépi látáson alapuló mapping és tracking eljárások egyik nagy problémája a sok adat tárolása, ami leginkább az egyes pontfelhők méretéből adódik. Így néhány módszer csak kisebb, szobaléptékű rekonstrukcióra képes [10]. Sok kutatás erőssége, hogy skálázható, így nagy mennyiségű adat kezelésére alkalmas.

2.5. Struktúrált megvilágítás

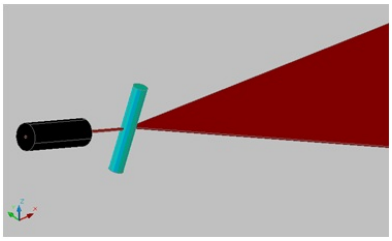
Kutatásom során megvizsgáltam egy strukturált megvilágításon alapuló saját fejlesztésű szenzort [24], amit egy kis méretű robotautóra fel lehet helyezni és kis erőforrás igényű. A vizsgálat során lokális és globális térképet állítottam elő a szenzor segítségével a robot autonóm navigációjához. A következőkben ennek a rendszernek a bemutatása történik.

2.5.1. Szenzor

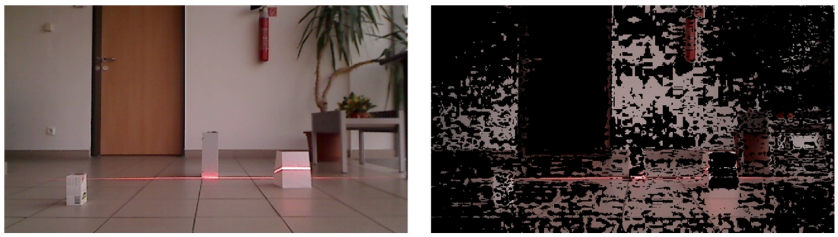
A lézer alapú strukturált megvilágítási technikáknak három alapvető típusa van [58]:

- trianguláris szenzoros (lézeres, Kinect, . . .)
- fázismodulációs szenzoros,
- TOF kamerás.

A szenzorrendszerem két részből tevődik össze, ami egy LEGO toronyra lett felhejezve. Az első része egy 150 mW-os vörös fényt kibocsájtó lézerprojektor. Ennek a kivetült fényét



2.1. ábra. A szenzor optikája



2.2. ábra. Piros színű pixelek a HSV színtérben szűrve a HUE 355 - 10 tartományon

egy henger alakú optika vonlaszerűvé formálja (2.1. ábra). A másik rész egy Logitech V-UBM46 webkamera, ami roboton helyezkedik el és érzékeli a robot elé kivetített lézerfényt.

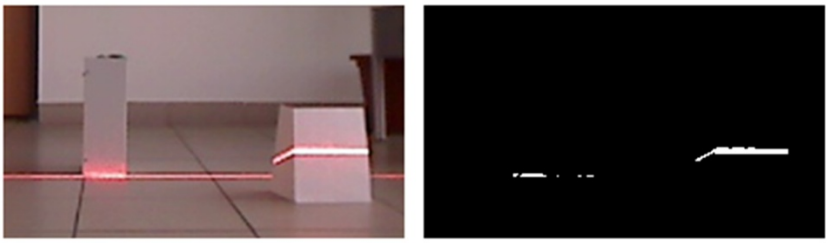
A vizsgált tartomány méretét a kamera látószöge és a kamera tengelye, valamint lézer által bezárt szög együttesen határozza meg. Ha a szenzor előtt akadály van, a lézer fénye arról verődik vissza, így ezen a területen a lézer a kamera által leképezett képen magasabban helyezkedik el. Ebből az eltérésből lehet az akadály távolságát kiszámítani.

2.5.2. Lézerdetektálás

A detektálás feladata a lézercsík meghatározása a képen a zavaró hatások kiküszöbölése mellett.

Színtér alapú lézerdetektálás

A színszegmentáláson alapuló szűrők a lézer diszkrét spektrumú sugárzását használják ki. Az objektumról visszaverődő lézercsík hullámhossza nem változik a beeső fény erősségének és beesési szögének változásával, csupán az intenzitása. Így elegendő a piros hullámhosszú visszavert fénysugarak vizsgálata. A lézer piros színének szűrése többek között RGB és HSV színtérben lehetséges. (2.2. ábra).



2.3. ábra. Intenzitás alapú lézerdetektálás, 250-es küszöbérték mellett

Intenzitás alapú detektálás

A magas intenzitású piros pixelek keresésénél egy forráskép minden pixelén egy transzformációt (2.39) végrehajtva egy másik kép generálódik (2.3. ábra).

$$g(x,y) = \begin{cases} 0 & \text{if } f(x,y) < k, \\ f(x,y) & \text{if } f(x,y) \geq k. \end{cases} \tag{2.39}$$

ahol a forráskép „f”, a célkép „g”, x és y a képpont koordinátái és a küszöbérték k.

A módszer problémája, hogy csak sötétebb környezetben hatásos, valamint minden magas intenzitású területről feltételezi, hogy az egy keresett terület.

Súlyozásos módszer

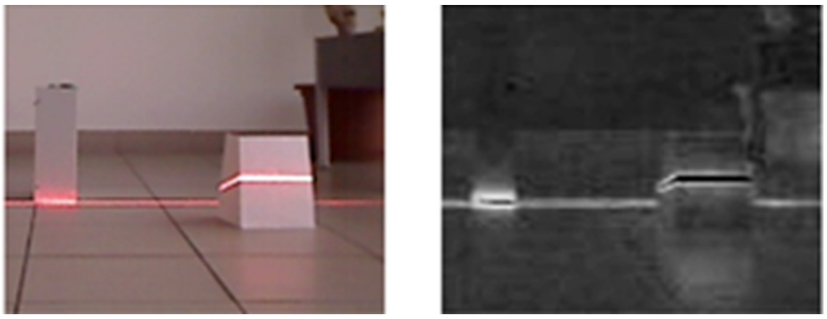
A fehér felületek zavaró hatásának kiküszöbölésére az egyik módszer, hogy képalkotásnál az RGB színcsatornából a piros csatorna kétszeres értékéből a zöld és a kék értékeket kivonja. Ennek eredményeképp olyan képet hoz létre, melyben csak a piros árnyalatú pixelek maradnak meg, a (2.40) és (2.41) kifejezések szerint:

$$g(x,y) = 2I_r(x,y) - I_g(x,y) - I_b(x,y) \tag{2.40}$$

$$h(x,y) = \begin{cases} 0 & \text{if } g(x,y) < 0, \\ g(x,y) & \text{if } 0 \leq g(x,y) \leq 1, \\ 1 & \text{if } \textit{other}. \end{cases} \tag{2.41}$$

ahol I_r a piros, I_g a zöld, I_b pedig a kék színcsatorna intenzitás értéke, a maximális intenzitás értéke 1, g átmeneti kép, h pedig a célkép.

Itt problémát okoz, hogy a túl nagy intenzitású lézercsík beég a képbe, melyet az eljárás „eldob”, így hasznos információt vesztünk (2.4. ábra).



2.4. ábra. Sújozott szűrő eredménye

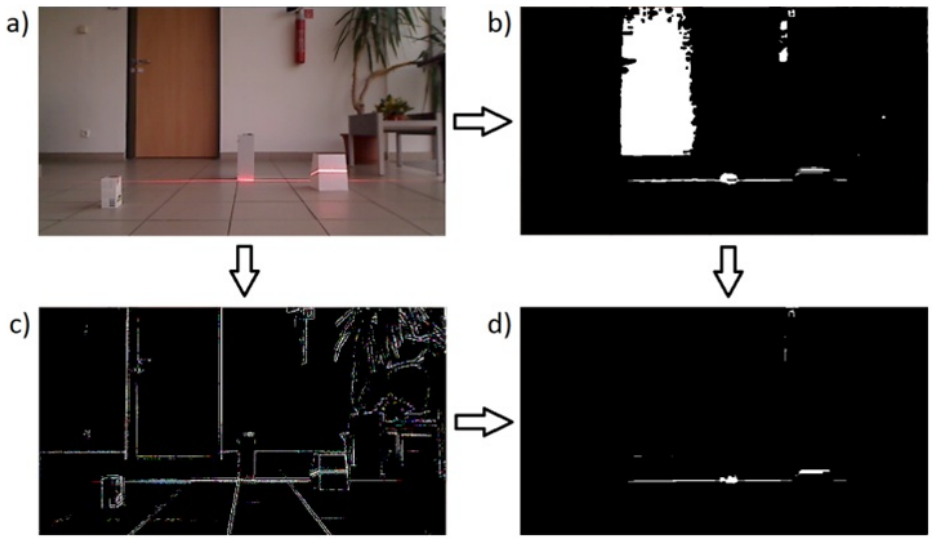
2.5. ábra. A LoG éldetektálás eredménye (küszöbérték: 150)

Éldetektálás alapú lézerdetektálás

A lézercsík elhelyezkedése éldetektáló algoritmus segítségével is kinyerhető a képből. A LoG (Laplacian of Gaussian) éldetektálás maszk alapú éldetektáló. Az élek irányultságát nem veszi figyelembe és Gauss zajszűrést is tartalmaz. Probléma ezzel a módszerrel, hogy egy lézercsíkot több élnek feleltet meg [59]. A Canny éldetektálás az éleket több lépésben határozza meg [60]. Minden élpontot csak egyszer jelez, így az éleket pontosan lokalizálja. Nem zajérzékeny eljárás, és minden valódi élt detektál. Valós idejű képfeldolgozásnál hátrányt okozhat, hogy az algoritmus jóval lassabb, mint az egyszerű, konvolúciós mátrixon alapuló élkereső eljárások (2.5. ábra).

Több módszer kombinálása

A korábbi pontokban ismertetett módszerek kombinálásával egy jobb minőségű detektálás valósítható meg (2.6. ábra). A súlyozásos és az intenzitás alapú keresés összevonásának



2.6. ábra. Lézerdetektálás; a) bemeneti kép, b) súlyozott szűrő, c) Laplace éldetektálás d) végső kép (”b” és ”c” kép logikai ÉS kapcsolata)

(2.42) eredményéül a magas intenzitású, piros színtartományba eső pixelek maradnak meg (2.6.b ábra). Ez kiküszöböli a lézercsík beégése miatt keletkezett információveszteséget.

$$q(x,y) = max(h(x,y), g(x,y)), \tag{2.42}$$

ahol $q(x,y)$ az eredménykép, $h(x,y)$ a súlyozott kép, $g(x,y)$ pedig az intenzitás alapú kép pontjai.

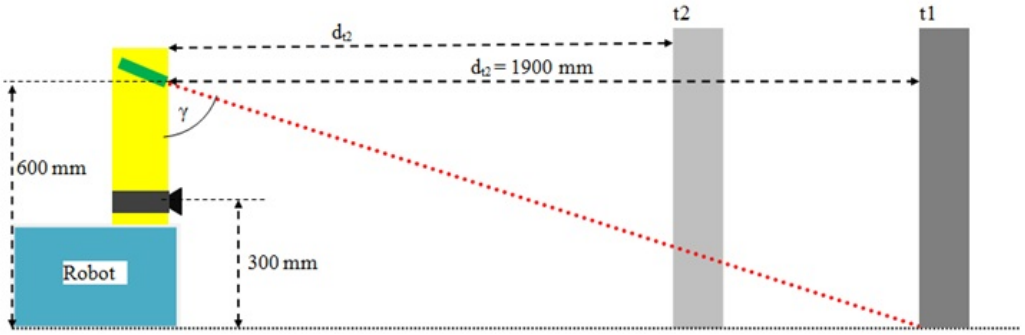
A 2.6.c ábrán látható a Laplace éldetektáló eredménykép. Ebben az esetben a detektálás nagyon sok hibát tartalmaz. Azonban a két eredménykép logikai „és” kapcsolatát véve (2.43), eredményül a kívánt jel megmarad.

$$z(x,y) = q(x,y)\ AND\ l(x,y), \tag{2.43}$$

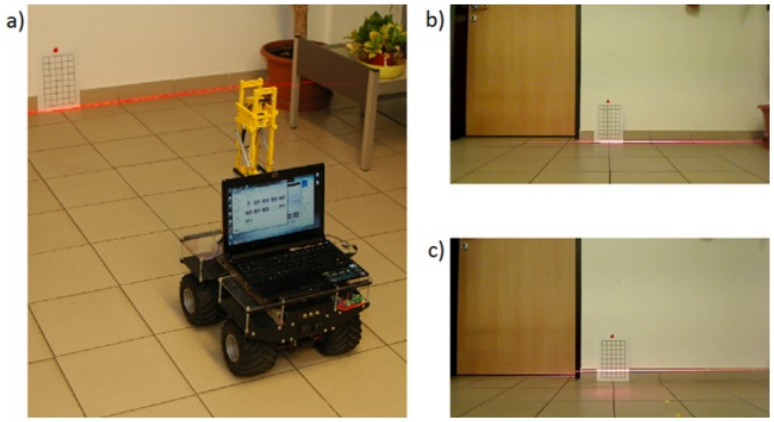
ahol $q(x,y)$ az infraszűrő és a magas intenzitású pixelek által szolgáltatott képek összegének pontjai és $l(x,y)$ a Laplace éldetektáló által szolgáltatott kép. A detektált jellemzőt a 2.6.d kép mutatja.

2.5.3. Mélységi térkép

A lézer a kamera síkjára nem merőlegesen, hanem ahhoz képest γ (90 foknál kisebb) szögben helyezkedik el, mint azt a 2.7. ábra is szemlélteti. Így ha egy tárgyon a lézer fénye megtörik ez a kapott képen egy bizonyos magasságban jelenik meg. Ez látható a



2.7. ábra. Lézerszenzoros rendszer felépítése

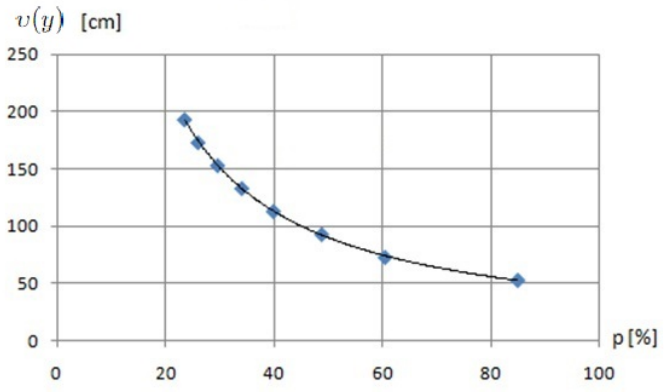


2.8. ábra. Szenzor kalibráció. a) a mérési környezet, b) amikor a jármű és az akadály távolsága $d_{t1} = 1,9\ m$, c) amikor a távolság $d_{t2} = 1,7\ m$

2.8. képen. A ”b” és ”c” képen látható, ahogy a jármű közeledik egy tárgyhoz, a róla visszaverődő lézer képe egyre magasabban helyezkedik el a kapott képen.

Első lépés a szenzor kalibráció volt. Ennek első elemeként a legtávolabbi pont mérése történt meg, amikor a mérőjel a földre vetült. A legtávolabbi pont 1900 mm-re volt mérhető az adott beállításnál. Majd a mérést 20 centiméterenként megismételve, az autó egyre közelebb került a falhoz, miközben feljegyzésre került a lézercsík elhelyezkedése a képen p_y (ez a mérőszám a kép aljától a teteje fele növekszik 0..100 tartományon).

Ezen mérések ismeretében meghatározható a tárgytól való távolság, a lézercsík elhelyezkedésének függvényében, amit a 2.9. ábra szemléltet. A ponthalmazra függvény illesztésével meghatározható a távolság és a lézerpozíció közti összefüggés. A mérésekből az y-odik helyzetben a távolság értéke milliméterben kifejezve (2.44).



2.9. ábra. A lézerpozíció és a távolság kapcsolata

$$\upsilon(y) = 47366 * p_y^{-1,013} \tag{2.44}$$

Térkép készítése

Az útvonaltervezéshez szükség van mélységi térképre. Ez egy kétdimenziós $D[r, e]$ mátrix, ami megadja a jármű előtt elhelyezkedő tárgyak térbeli helyét (2.10. ábra). A mátrix minden eleme a tér egy kis részletét reprezentálja. A térkép jól szemlélteti, hogy a különböző objektumok, milyen távolságra, és milyen irányban helyezkednek el a járműhöz képest.

A mátrix $D_{0,0}$ pontja az alsó sor középső eleme. Ez a pont a jármű helyzetét mutatja. Ezen ponttól távolodva a mátrix sorain haladva, a járművel szemben elhelyezkedő tárgyak távolsága adható meg. Minél nagyobb számú sorban helyezkedik el egy érték, az objektum annál messzebb helyezkedik el. A mátrix oszlopain, a középső oszloptól távolodva, jobbra, illetve balra haladva, a jármű mozgástengelyére merőlegesen adja meg az objektumok távolságát.

A mátrix elemei közötti távolság $\Delta e = e_n - e_{n+1}$ és $\Delta r = r_n - r_{n+1}$ teremt kapcsolatot a valós távolságok között. A kamera által szolgáltatott képből, elkészült bináris kép (z), ami már csak a lézercsíkot tartalmazza, minden értéket tartalmazó pontja, egy az autó előtt elhelyezkedő objektumról ad információt. Az előzőleg meghatározott kameraparaméterek ismeretében megadható $upsilon(y)$ függvény (2.44), ami a képen a lézer magasságához, hozzárendeli a tárgy távolságát. Vagyis a mélységi mátrix hányadik sorához tartozik az objektum, valamint egy $\varphi(x)$ függvény (2.45) ami az autóhoz képest az oldalirányú tárgytávolságot adja meg. A $\varphi(x, y)$ függvény a mátrix középső oszlopához képest $D_{0,0}$

$$\begin{bmatrix} D_{r-1,-(e-1)/2} & \cdots & D_{r-1,-1} & D_{r-1,0} & D_{r-1,1} & \cdots & D_{r-1,(e-1)/2} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ D_{1,-(e-1)/2} & \cdots & D_{1,-1} & D_{1,0} & D_{1,1} & \cdots & D_{1,(e-1)/2} \\ D_{0,-(e-1)/2} & \cdots & D_{0,-1} & D_{0,0} & D_{0,1} & \cdots & D_{0,(e-1)/2} \end{bmatrix}$$

2.10. ábra. Mélységi térkép reprezentációja

adja meg a távolságot.

$$\varphi(x, y) = \frac{x}{0,0447 * y - 1,4267}, \tag{2.45}$$

ahol az y a detektált lézerkép (z) sorát adja meg, maximális értéke h. Valamint x a kép oszlopát tartalmazza, maximális értéke w. Jelen számításoknál $h = 360$ és $w = 640$. Az 2. algoritmus megadja a forráskép pontjainak hozzárendelését a mélységi térképhez.

2. algoritmus Mélységi térkép készítése

Input: w, h a kép szélessége és magassága, egészek
Input: $v(j)$ a lézer magasságához hozzárendeli a tárgy távolságát
Input: $\varphi((i - \frac{w}{2}, j)$ az oldalirányú távolságot adja
Output: $D[]$ mélységi térkép tömbje

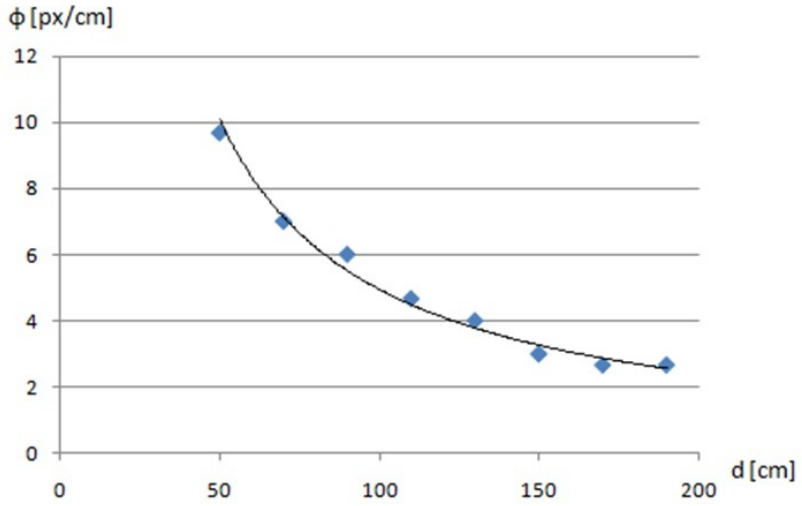
```
1: for i = 0 → h do
2:     for j = 0 → w do
3:         if 0 ≤ v(j) < r AND −e/2 < φ((i − w/2), j) < e/2 then
4:             D[v(j), φ((i − w/2), j)] = 1
```

A megvalósított szoftver a mátrixot képként tárolja. A mátrix egy elemét, egy pixelnek, felelteti meg. A szomszédos pixelek távolsága 1 cm. Az akadályok elhelyezkedése a korábban kapott fekete képen a fehér pixelek helyzetének feletethető meg. Így előáll a mélységi térkép.

Pontosság

Egy tárgy távolsága befolyásolja a mérési pontosságot, hiszen minél messzebb helyezkedik el, annál kisebbnek látszódik. A valós tárgy leképezésének arányát a 2.11 ábra szemlélteti. A távolság növekedésével egy centimétert egyre kevesebb pixel ábrázol. Így a lézercsík pontos azonosítása a képen egyre nagyobb bizonytalansággal történik meg.

A táblázatból jól látható (2.1. táblázat), hogy a maximális hibaértékek még távoli tárgyaknál sem túl nagy, közeli tárgyaknál pedig a mérési hiba csökken, vagyis a mérés pontossága függ a tárgyak távolságától. A jármű haladása közben a bejárandó útvonal



2.11. ábra. Struktúrált megvilágítást használó szenzor pontossága a távolság függvényében. Az ábrán látható, hogy egy pixel a képen hány cm-t reprezentál az objektum távolságának függvényében, ahol d az akadály szenzortól mért távosága és ϕ hány pixel felel meg 1 cm-nek az adott távolságban.

egyenetlenségeiből adódóan a jármű kis mértékben előre vagy oldalra megdőlhet.Ezért a felhasználás során törekedtünk, hogy a jármű előre-hátra dőlése minél kisebb legyen, hiszen az a pontosságot rontja. A hosszanti tengely körüli dőlést is kerültük a felhasználás során, amit alacsony sebesség beállításával biztosítottunk.

2.6. RGB-D kamera

A Kinect egy RGB-D szenzor, amit a Microsoft játékvezérlésre fejlesztett ki 2010 végén [7]. A cél az volt, hogy a játékok vezérléséhez ne legyen szükség joystick-ra, billentyűzetre, egérre. Az elsőnek kiadott kamera (2.12. ábra) verzió 640 × 480-as felbontású színes képet ad (ami a későbbi verziók esetén nagyobb lett), amihez pixelszintű távolsági adatok tartoznak. A mélységi szenzor érzékelési tartománya 0,5 méter és 4 méter között van (Windows SDK esetén). A mélységi kamera pontossága ezen tartományon körülbelül 10–60 mm között változik a saját mérések alapján.

A Kinect a sztereókamerákkal ellentétben a távolság mérésére egy aktív mérőjelet vetít ki. Az aktív mérőjelet használva a szenzornak nincs szüksége külső fényforrásra a működéshez. A szenzor nagy külső megvilágítás esetén, például kültérben napsütés esetén nem képes érzékelni a saját mérőjelét, így csak a beltéri használat esetén ad megbízható

Valós érték	Mért érték	Relatív hiba
50 cm	50 cm	0 %
100 cm	99 cm	1 %
150 cm	145 cm	3,3 %
170 cm	165 cm	3 %

2.1. táblázat. Struktúrált megvilágítást használó szenzor hibája, különböző távolságok esetén.

mérési eredményeket. Valamint az aktív fényforrással rendelkező tárgyakat, így például a lámpákat sem képes érzékelni (2.13. ábra). A 4 méteres érzékelési tartomány alkalmassá teszi a szenzort egy kisebb méretű roboton való használatra.

A fejezet további részében a Kinect szenzor pontossága, kalibrációja és adatainak feldolgozása kerül vizsgálatra. Ehhez a következő jelölésrendszer kerül bevezetésre.

- $[u_d, v_d]^T$, mélységi kamera pixelkoordinátája, és $\delta(u_d, v_d)$ a pixel mélységi értéke.
- $[u_c, v_c]^T$, színes kamera pixelkoordinátája.
- $p_d = [x_d, y_d, z_d]^T$, mélységi kamera koordináta-rendszerében ábrázolt háromdimenziós pont.
- $p_c = [x_c, y_c, z_c]^T$, színes kamera koordináta-rendszerében ábrázolt háromdimenziós pont.
- $p_w = [x_w, y_w, z_w]^T$, világ koordináta-rendszerben ábrázolt háromdimenziós pont.

2.12. ábra. Kinect kamera

2.13. ábra. Kinect 1-es szenzor kimenete aktív fényforrás esetén. Az alsó képen a szenzor által szolgáltatott adatok képként jelennek meg. Feketével, ahol nincs távolsági adat az aktív fényforrás (vagy reflexió) miatt.

- $X_d = \{p_{d_1}, p_{d_2}, ..., p_{d_n}\}$, mélységi kamera koordináta-rendszerében ábrázolt pontok halmaza, ahol n nem nagyobb mint a mélységi kép pixeleinek száma.
- $X_w = \{p_{w_1}, p_{w_2}, ..., p_{w_n}\}$, világ koordináta-rendszerben ábrázolt pontok halmaza.
- $f_c = [f_{cx}, f_{cy}]$, $p_c = [p_{cx}, p_{cy}]$, színes kamera fókusztávolsága (focal length) és optikai középpontja (principal point).
- $f_d = [f_{dx}, f_{dy}]$, $p_d = [p_{dx}, p_{dy}]$, mélységi kamera fókusztávolság (focal length) és optikai középpontja (principal point).
- $\tau_{cd} = \{T_{cd}, R_{cd}\}$, a színes kamera és mélységi kamera koordináta-rendszere közötti transzformáció.

A Kinect gyárilag kalibrálva van, amely a fedélzeten van tárolva egy magas szintű polinomiális torzítási függvény alapján. Az OpenNI illesztőprogram ezt a kalibrációt használja a képek torzításmentesítésére és a mélységi képek (amelyeket az IR kamera készít) RGB képekkel való regisztrálására. Ezért a mélységi képek át vannak vetítve a színes kamera koordináta-rendszerébe, ami azt jelenti, hogy a mélységi térképen és a színes képen lévő pixelek között 1:1 megfeleltetés van.

A 2D képekből 3D pontfelhőkbe való átalakítás a következőképpen működik. Megjegyzendő, hogy a fókusztávolságok (f_x/f_y), az optikai középpont (c_x/c_y), a torzítási paraméterek ($d_0 - d_4$) és a mélységi korrekciós tényező kameránként eltérőek. A

3. algoritmus bemutatja, hogyan számítható ki a 3D pont a pixelek koordinátáiból és a mélységi értékből. Meghatározza a 3D koordinátákat (X, Y, Z), ahol a távolsági képeknél alkalmazott faktor (16 bites PNG fájlok esetén):

$$factor = 5000,$$

és a távolságok:

$$Z = \frac{\text{depth\_image}[v,u]}{factor}$$

$$X = \frac{(u - c_x) \cdot Z}{f_x}$$

$$Y = \frac{(v - c_y) \cdot Z}{f_y}$$

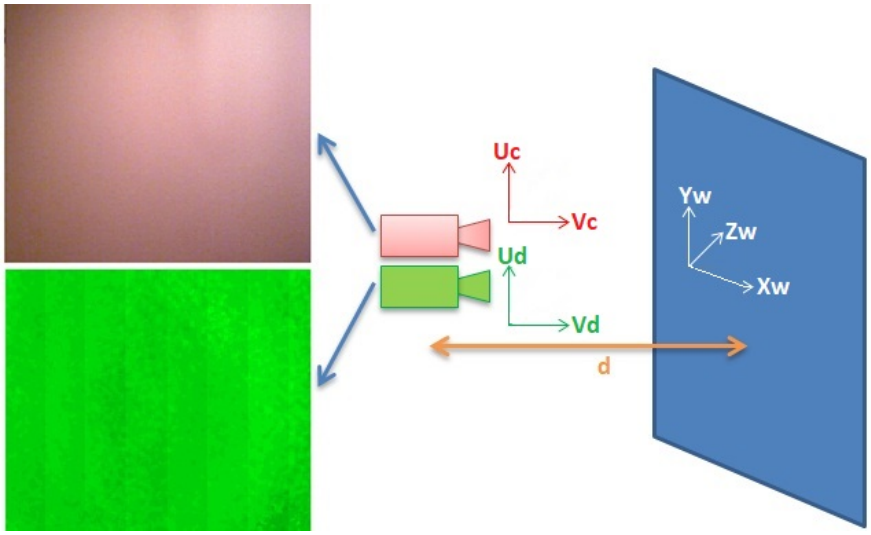
3. algoritmus Az RGB-D szenzor mérési értékeiből a 3D koordináták meghatározása

Input: f_x, f_y a színes kamera fókusztávolsága
Input: c_x, c_y a színes kamera optikai középpontjának koordinátái
Input: depth_image[v,u] a mélységi kamera képe
Input: $factor$ a távolsági képeknél alkalmazott faktor, 16 bites PNG fájlok esetén 5000)
Output: X, Y, Z 3D koordináták

1: **for** $v = 1 \rightarrow$ depth_image.height **do**
2: **for** $u = 1 \rightarrow$ depth_image.width **do**
3: $Z = \frac{\text{depth\_image}[v,u]}{\text{factor}}$
4: $X = \frac{(u - c_x) \cdot Z}{f_x}$
5: $Y = \frac{(v - c_y) \cdot Z}{f_y}$

Herrera és kollégái is publikáltak egy kalibrációs eljárást a Kinect szenzorhoz, ami meghatározza a kamera belső paramétereit és a kamerák koordináta-rendszere közti transzformációkat [61]. A cikkben a mélységi kamera távolság adatainak pontosságával nem foglalkoznak, ezért szükséges volt a szenzor mélységi kamera pontosságának meghatározása.

A mélységi kamera pontosságának meghatározására egy tesztkörnyezet került felállításra. A szenzort egy sík felületre merőlegesen állítva, különböző d_i távolságok esetén készültek mélységi képek a sík felületről. A mélységi képen a valós távolságtól való eltérések adják



2.14. ábra. Tesztkörnyezet a mélységi kamera pontosságának meghatározására. A tesztkörnyezet $1m$ távolság esetén készült szines és mélységi képe látható az ábrán. Kék négyzet a kamera tengelyére merőleges sík felület, d távolság esetén.

a mérési hibákat. A mérési környezet a 2.14. ábrán látható. A 2.15. ábra, baloldali képén látható $1m$ távolságnál készült távolsághisztogram.

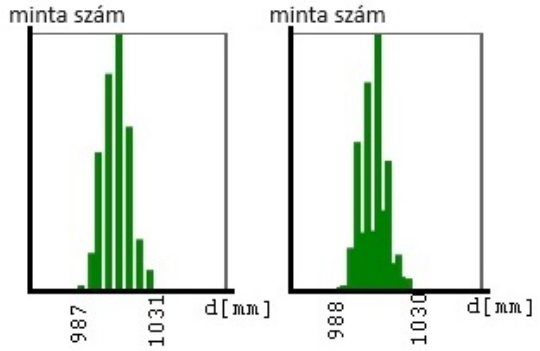
2.6.1. Az RGB-D szenzor legnagyobb mérési hibája

A szenzor pontossága két mérőszámmal került jellemzésre. Az egyik a Δd (2.46) a valós távolságtól mért legnagyobb eltérés a képen. Ez megadja a legnagyobb mérési hibát, amit a Kinect szenzor adhat. A kisebb mérési zajok kiszűrése érdekében a mélységi kép összes pontjára és a legrosszabb eredményét adó 1% pont elhagyása esetén is meghatározásra került a Δd.

$$\Delta d = max(|d - min(\delta(u_d, v_d))|, |d - max(\delta(u_d, v_d))|) \tag{2.46}$$

A mélységi képek hisztogramjain jól látszik, hogy a távolságok a valós értékhez közelítenek, de elég nagy zajjal rendelkeznek, ami pontatlan eredményeket ad. A mérési zaj csökkentésére egy feltételes átlagoló szűrőt alkalmaztam a mélységi képen. Az algoritmus (4. algoritmus) egy 3×3-as átlagoló szűrőn alapszik, de kihagyja azokat a pontokat, ahol a távolságmérés hibás eredményt adott, vagyis értéke nulla [28] [36].

A feltételes átlagoló szűrő után kapott eredménykép hisztogramját a 2.15. ábra jobb oldali képe mutatja. A legnagyobb mérési hibákat a szűrő nem képes megszüntetni, de a

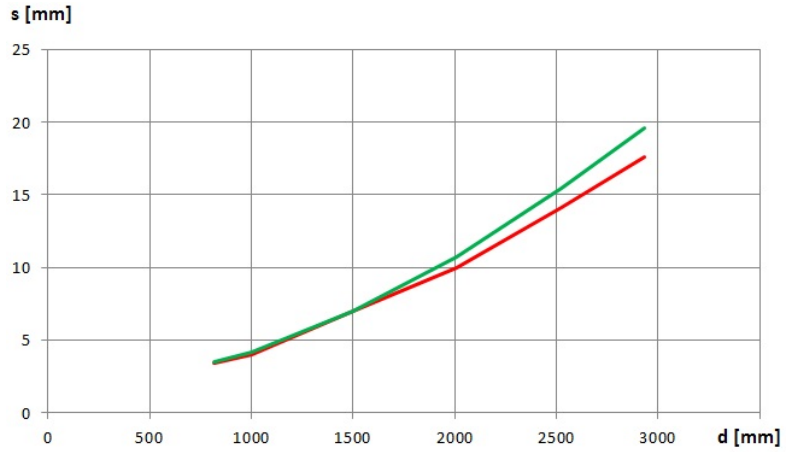


2.15. ábra. Távolsághisztogram, a vízszintes tengelyen a mélységi képen lévő diszkrét távolságértékek, a függőleges tengelyen az adott távolságra mért pontok száma. Az eredeti mélységi kép (baloldali kép), és a mélységi adatok feltételes átlagoló szűrése (jobboldali kép) utáni hisztogramok láthatóak az ábrán.

4. algoritmus Mélységi kép szűrő algoritmusa (DIF)

Input: w, h a mélységi kép szélessége és magassága
Input: δ a szűrendő mélységi kép
Output: δ' a szűrt mélységi kép

for $i = 1 \rightarrow (w-1)$ **do**
  for $j = 1 \rightarrow (h-1)$ **do**
    $k = 0$
    $T_k = 0$
    for $m = i-1 \rightarrow i+1$ **do**
      for $n = j-1 \rightarrow j+1$ **do**
        if $\delta(m,n) \neq 0$ **then**
          $T_k = T_k + \delta((m,n)$
          $k = k+1$
    if $k \neq 0$ **then**
      $\delta'(i,j) = (\sum_{l=1}^{k} T_l)/k$



2.16. ábra. A Kinect szenzor távolságmérésre adott szórása az összes mérési pontra zöld színnel látható. A piros színű függvény a szenzoradatok átlagoló szűrő használata utáni szórását szemlélteti.

kisebb egyenetlenségeket lesimítja.

2.6.2. Az RGB-D szenzor szórása

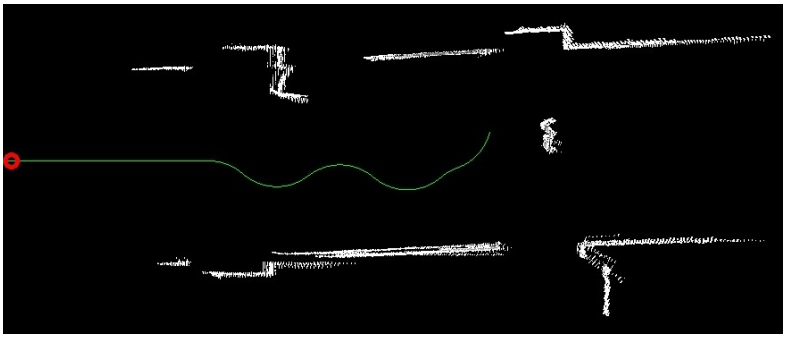
A szenzor pontosságára a másik mérőszámot a mélységi kép szórása s adja. A 2.16. ábra a szórás távolság függvényt mutatja, zöld színnel az összes pontra vett szórás. Piros színnel a feltételes átlagoló szűrő alkalmazása után kapott szórás látható az ábrán. Az átlagoló szűrő a szenzor szórását csökkentette.

2.6.3. Térképépítés

A következőkben a térképépítés témakörét tárgyalom, azaz miként lehet a robot által bejárt terület háromdimenziós térképét felépíteni, ismerve a szenzor pontosságát és belső paramétereit. Egy korábbi cikk [27] során foglalkoztam a Kinect szenzor adataiból történő kétdimenziós térkép építéssel. A lokális térképek illesztésékez a roboton elhelyezett kerék enkóderek adtak illesztési információt. A távadók a kétdimenziós odometriát viszonylag nagy pontossággal képesek megadni két egymás utáni szenzoradat között. Az egyik tesztelés során készült térképet a 2.17. ábra mutatja.

A megvalósításban csak a mélységi adatok kerültek felhasználásra és csak kétdimenziós térképet készített a rendszer. A következő cél, hogy a rendszer háromdimenziós térképet építsen fel a környezetéről, és ehhez ne csak az odometriai adatokat használja.

A háromdimenziós globális térkép építésének egyik feltétele, hogy ismert legyen a szenzor körüli tér háromdimenziós képe egy adott pillanatban. A (2.47.) képlet megadja a



2.17. ábra. Kétdimenziós globális térkép, egy korábbi munkámban [27]. Fehér színnel az objektumok, zöld vonal a robot által bejárt útvonal, a piros kör a kiindulási helyzetet mutatja.

mélységi kép egy pixelének háromdimenziós helyzetét (p_d). Ha egy mélységi kép minden képpontján végrehajtásra kerül ez a transzformáció, eredménye az X_d halmaz lesz, ami egy háromdimenziós pontfelhő a szenzor aktuális környezetéről.

$$p_d = \begin{bmatrix} (u_d - p_{dx}) * \delta(u_d, v_d)/f_{dx} \\ (v_d - p_{dy}) * \delta(u_d, v_d)/f_{dy} \\ \delta(u_d, v_d) \end{bmatrix} \tag{2.47}$$

A rendszer működése során sorra készülnek mérések a Kinect szenzorral, így több időben egymást követő pontfelhőt kapunk eredményül ($X_{d1}, X_{d2}, ..., X_{dn}$). A végső cél, hogy az egyes pontfelhőket egy nagy globális térbe legyen képes illeszteni a rendszer (X_w). Ehhez szükséges ismerni, hogy az egyes mérések között milyen mértékben mozdult el a szenzor. Az elmozdulás egy első becslését az odometriából kaphatjuk, amit már a korábbi munka esetében is felhasználtam. Az inkrementális érzékelők esetén csak a robot sík felületen való mozgása követhető.

$$[u'_c, v'_c]^T = \begin{bmatrix} f_{cx} & 0 & 0 \\ 0 & f_{cy} & 0 \end{bmatrix} * p'_d/\delta(u_d, v_d) + p_c, \tag{2.48}$$

ahol az elmozdult pont helye $p'_d = R_{cd} * p_d + T_{cd}$, R_{cd} a Kinect szines és mélységi kamera közti rotációs és T_{cd} a Kinect szines és mélységi kamera közti transzlációs mátrixa.

További pontosítást adhat a transzformáció meghatározására, amennyiben felhasználásra kerül a szenzorból kinyert vizuális információ is. A mélységi képekhez tartozó színes képen jellemzőpontok keresésével lehetőség van a háromdimenziós pontfelhőben kijelölni jellemző háromdimenziós pontokat. Ehhez szükséges még ismerni a mélységi kamera és a

szines kamera közötti pixelszintű kapcsolatot, amit a (2.48) képlet ad meg. Az egymás utáni képeken nagy valószínűséggel lesz néhány olyan jellemzőpont, ami mindkét képen megtalálható, így ezek között már meghatározható a szükséges transzformáció. Ennek a módszernek a bemutatása a disszertáció 4. fejezetében történik.

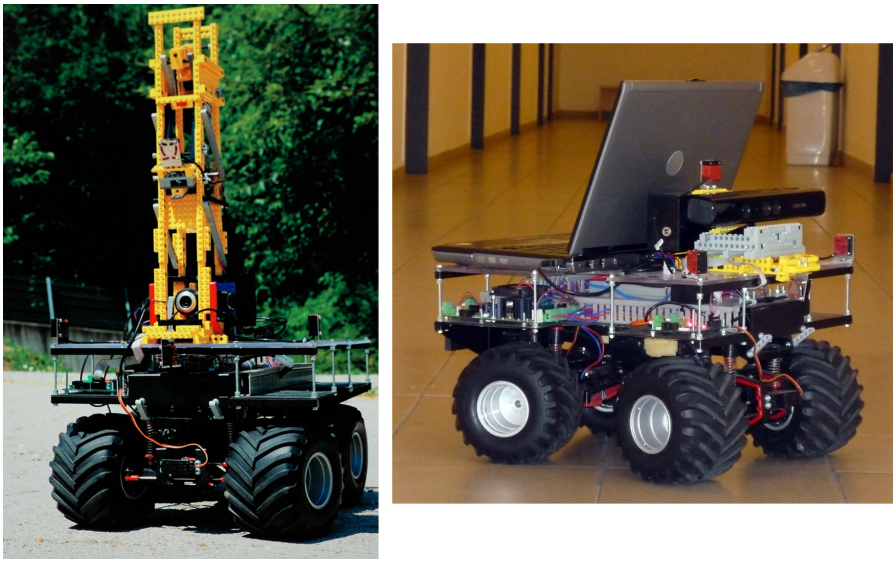
3 Tesztrendszer építése, robotautó

3.1. Adatgyűjtő rendszer építése

Az egyes szenzorok és algoritmusok kipróbálására készült egy saját robotautó. Az autóra az éppen tesztelt szenzorok felszerelésre kerültek. A vezérlő szoftver tesztelése a járműre felhelyezett laptopon is képes futni [24].

3.1.1. Robotautó

A robot alapja egy 1/8-os méretarányú RC elektromos távirányítós autó. A jármű mindkét tengelye elektronikus hajtással rendelkezik, és első és hátsó kerekei szervomotor segítségével kormányozhatóak. Az autó viszonylag kis méretű, ami beltéri navigációra is alkalmassá teszi, de kisebb szenzorokat és egy laptopot képes hordozni. Az 3.1. ábrán látható a robotautóról készült kép.

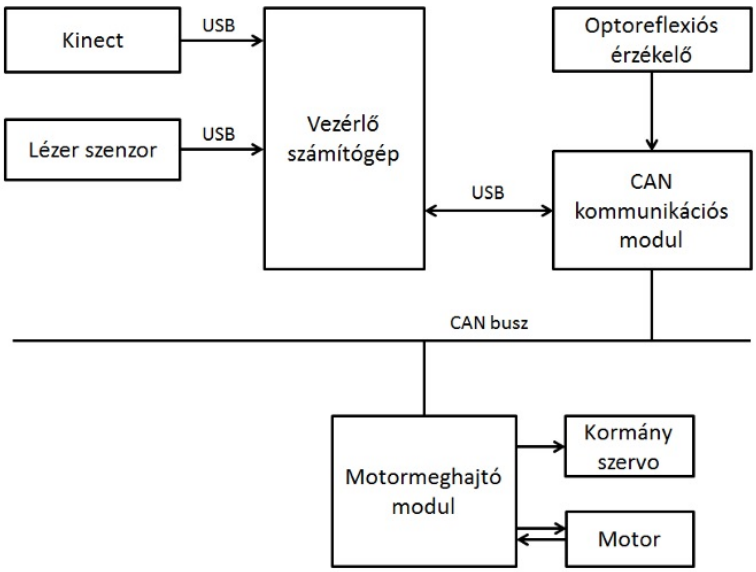


3.1. ábra. Robotautó, baloldali képen lézerszenzor [24], a jobboldali képen Kinect szenzor látható rajta [27]

A roboton két fő elektronikai egység található (3.3. ábra). Ezek az elektronikák CAN

buszon kommunikálnak egymással (3.2. ábra). A CAN busznak köszönhetően lehetőség van újabb vezérlőegységeket csatolni a rendszerhez. A rendszerben egy általam tervezett motormeghatjó és egy CAN kommunikációs elektronika van.

A CAN bridge kártya (6.1.3. melléklet) elsődleges feladata, a CAN üzenetek továbbítása a számítógép felé. A modul virtuális soros portonkeresztül kapcsolódik a PC-hez, egy FTDI IC-vel. Az áramkör tápellátása széles feszültségtartományon történhet. A modulra lehetőség van LCD kijelzőt csatlakoztatni, amin a rendszer állapota jeleníthető meg, mint például az akkumulátorok töltöttsége.

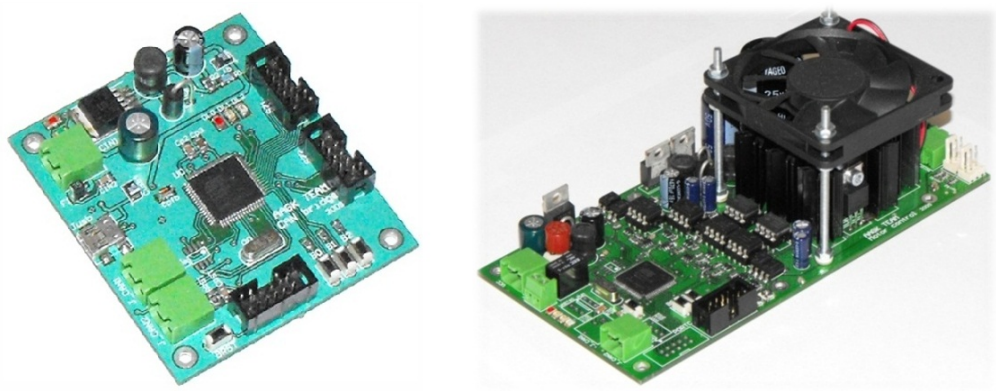


3.2. ábra. A robotautó rendszerének felépítése

A motormeghajtó modul (6.1.3. melléklet) a jármű mozgásáért felel. Gondoskodni kell a jármű előre, hátra történő mozgatásának és a haladási irányának megváltoztatásáról. A jármű kerekére szerelt, inkrementális szenzorral határozható meg az egységnyi Δt idő alatt megtett út (Δl). A mérés pontosságát a talajminőség befolyásolhatja [62], ezért a rendszer jelenleg csak sík, egyenletes talajfelületen működik megfelelő pontossággal [23].

3.2. Tesztalkalmazás

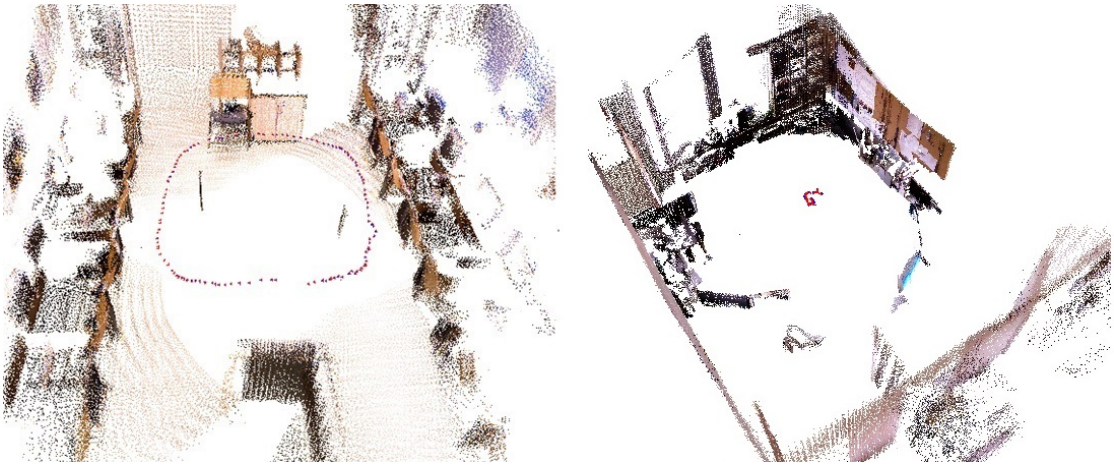
Az elkészült alkalmazás tervezésénél egyik fő szempont az volt, hogy a rendszer képes legyen különböző adathalmazokat kezelni. A program egy Kinect szenzor színes és mélységi adatait dolgozza fel. De a rendszer képes kezelni offline adatbázisból származó korábban letöltött adatokat is.



3.3. ábra. A robotautóra tervezett vezérlő elektronikák (6.1.3, 6.1.3 mellékletek)

Több, SLAM algoritmus teszteléséhez készült offline adatbázis érhető el az interneten. A jelenlegi program háromféle adatbázist tud egységes formátumura konvertálni, hogy a kifejlesztett algoritmus értelmezni tudja azokat:

- Müncheni Műszaki Egyetem (TUM) - fireiburg1, fireiburg2, fireiburg3 [33],
- Poznani Egyetem adathalmazok [34],
- Óbudai Egyetem, saját adathalmazok (3.4. ábra).



3.4. ábra. A rendszer által készített rekonstrukció a saját adathalmazokból. A bal oldalon egy egyetemi labor, jobb oldalon egy iroda részlete látható.

3.3. Saját adathalmaz készítése

Az algoritmus kiprobálására készítettem saját adathalmazokat is a Kinect szenzorral, a nyilvánosan elérhető adatbázisok mellé. A saját offline adathalmazokat az Óbudai Egyetemen, elsősorban a 2.13-as laborban rögzítettem (3.4. ábra). Egy másik offline adathalmazt egy 50m2-es lakás belsejéről is (5.4. ábra) felvettem. Ezek az adathalmazok nem rendelkeznek valós, abszolút térbeli pozíciókkal, de a labor esetében a kiindulási és a végpoziciók és orientációk azonosak voltak. Így vizsgálható, hogy a teljes bejárt útvonal alatt mennyi hiba halmozódott fel. A rögzítés során a Kinect szenzor színes és mélységi képeit, páronként mentettem le egy adatbázisba, amit később fel tudtam dolgozni offline.

3.4. Elérhető nyílt adathalmazok

A SLAM rendszerek kutatásával napjainkban egyre többen foglalkoznak. A kutatások nagy részénél használnak RGB-D kamerákat, így például a Kinectet. Két olyan offline adathalmazt kerestem az algoritmusom tesztelésére, aminél ezt a szenzort használták fel az adatgyűjtésre. Az egyik ilyen adatbázis a TUM [33], ami számos adathalmazt biztosít. Többféle Kinect szenzorral készült mérési sort tartalmaz, van olyan, amikor kézben mozgatták a szenzort (3.5. ábra), máskor egy robotautóra felszerelve gyűjtötték az adatokat. A tesztelés során használt másik külső adatbázist a Poznani egyetemen készítették egy laborban, és ez négy különböző adathalmazt tartalmaz [34]. Mindkét adatbázis minden adathalmaza rendelkezik minden kép külső szenzorrendszerrel rögzített térbeli pozicióival, így lehetséges az algoritmus pontosságának vizsgálata. Mivel ezeket az adatsorokat sokan haszálják, más rendszerek pontosságával is össze lehet hasonlítani a saját megoldás eredményeit.

3.5. Az egyes adathalmazok felépítése, egységes kezelése

Az egyes adathalmazokban különböző módon kerültek tárolásra a valós koordináták, a színes és mélységi képek, valamint a szenzorparaméterek.

A Müncheni Műszaki Egyetem (TUM) adathalmazát leíró fájl a következő formátumban tárolja soronként az egyes elmozdulásokat és orientációkat:

- *timestamp*: a kamerakép időbélyege, másodpercben megadva,
- $[x_w, y_w, z_w]^T$: az elmozdulás egy pontja, a külső mérőrendszer koordináta-rendszerében,

3.5. ábra. Baloldalt a TUM [33], jobboldalt a POZNAN [34] adathalmaz egy részlete.

- $[q_x, q_y, q_z, q_w]^T$: külső szenzorral rögzített orientáció értéke kvaternióban megadva, a külső mérőrendszer koordináta-rendszerében.

Ennél az adathalmaznál a mélységi és a színes kameraképek elnevezése az időbélyeg alapján történt és két külön mappába kerültek a képek (*/depth/timestamp.png* és */rgb/timestamp.png*).

A Poznani Egyetem adathalmazát leíró fájl a következő formátumban tárolja soronként az egyes elmozdulásokat és orientációkat:

- $ImageNumber$: a kamerakép sorszáma, 0-val kezdődően,
- $[x_w, y_w, z_w]^T$: az elmozdulás értéke a külső mérőrendszer koordináta-rendszerében,
- $[\phi_w, \theta_w, \psi_w]^T$: külső szenzorral rögzített orientáció értéke Euler-szögekkel megadva, a külső mérőrendszer koordináta-rendszerében.

A Poznani Egyetem adathalmaznál a mélységi és a színes kameraképek elnevezése a kép sorszáma alapján történt és két külön mappába kerültek a képek (*/depth/depth_0.png* és */rgb/rgb_0.png*).

Az egységes kezelés érdekében, egy "*mbag*" kiterjesztésű fájlt hoz létre elsőként az alkalmazásom minden adathalmazhoz. A fájl felépítése a következő.

- Forrás adatbázis típusa: $(!RAWLOG\_FB1)$
- Forrás adatbázis neve: $(!dbName : desk)$
- Soronként a szenzorok adatai:
  $(timestamp; rgbd; depth; x_w; y_w; z_w; q_x; q_y; q_z; q_w)$

– *timestamp*: a kamerakép időbélyege, másodpercben megadva,
– *rgbd*: színes kamerakép, ami a $[u_c, v_c]^T$ pontok halmazából áll,
– *depth*: mélységi kamerakép, ami a $[u_d, v_d]^T$ pontok halmazából áll,
– $[x_w, y_w, z_w]^T$: az elmozdulás és orientáció értéke a külső mérőrendszer koordináta-rendszerében,
– $[q_x, q_y, q_z, q_w]^T$: külső szenzorral rögzített orientáció és elmozdulás értéke kvaternióban megadva, a külső mérőrendszer koordináta-rendszerében.

Az adatokat egységes formátumúvá konvertálja a program. Minden egyes pozícióban készült méréshez egységes időbélyeget, mélységi és színes kamerakép elérési útvonalat, valós pozíciót (méterben), és kvaternióban megadott orientációt rendel. A Poznani Egyetem adathalmaza esetén az Euler-szögeket kvaternióra konvertálja a rendszer (3.5.1. fejezet). Az eredmények bemutatásánál a kvaternió értékek Euler-szögekké számítódnak át. Az alkalmazás, amennyiben offline adatokkal dolgozik, akkor ezeket az "*mbag*" fájlokat olvassa be minden egyes futás során.

3.5.1. Az Euler-szögekből kvaternióvá történő átalakítás, valamint kvaternió értékből az Euler-szögek meghatározása

A TUM adathalmaz esetén előfordul, hogy az Euler-szögekből kvaternióvá kell átalakítani a tárolt értékeket. Ilyenkor a következő módon járhatunk el. Legyenek az Euler-szögek:

$$\phi \quad \text{(Roll)}, \quad \theta \quad \text{(Pitch)}, \quad \psi \quad \text{(Yaw)}$$

A kvaternió komponensei ekkor:

$$q_w = \cos\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right) + \sin\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right) \tag{3.1}$$

$$q_x = \sin\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right) - \cos\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right) \tag{3.2}$$

$$q_y = \cos\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right) + \sin\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right) \tag{3.3}$$

$$q_z = \cos\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right) - \sin\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right) \tag{3.4}$$

Kvaternióból Euler-szögek számítása

Amennyiben adottak a kvaternió komponensei: q_w, q_x, q_y, q_z és az Euler-szögek (Roll, Pitch, Yaw) értékeit szeretnénk meghatározni, akkor a következő képpen kell eljárni.

$$\phi = \text{atan2}(2(q_w q_x + q_y q_z), 1 - 2(q_x^2 + q_y^2)) \tag{3.5}$$

$$\theta = \text{asin}(2(q_w q_y - q_z q_x)) \tag{3.6}$$

$$\psi = \text{atan2}(2(q_w q_z + q_x q_y), 1 - 2(q_y^2 + q_z^2)) \tag{3.7}$$

A speciális esetek akkor fordulnak elő, ha a Pitch szög (θ) eléri a $\pm\frac{\pi}{2}$ értéket, ami teljes tengelyirányú forgást eredményez. Ilyenkor a forgás a y tengely körül történik, amely 90 fokos vagy -90 fokos forgást eredményez.

- Ha $q_w q_y - q_z q_x = \frac{1}{2}$, akkor $\theta = \frac{\pi}{2}$ (90 fok):

  $$\theta = \frac{\pi}{2} \tag{3.8}$$

  Ebben az esetben:

  $$\phi = 2 \cdot \text{atan2}(q_x, q_w) \tag{3.9}$$

  és $\psi = 0$.

- Ha $q_w q_y - q_z q_x = -\frac{1}{2}$, akkor $\theta = -\frac{\pi}{2}$ (-90 fok):

  $$\theta = -\frac{\pi}{2} \tag{3.10}$$

  Ebben az esetben:

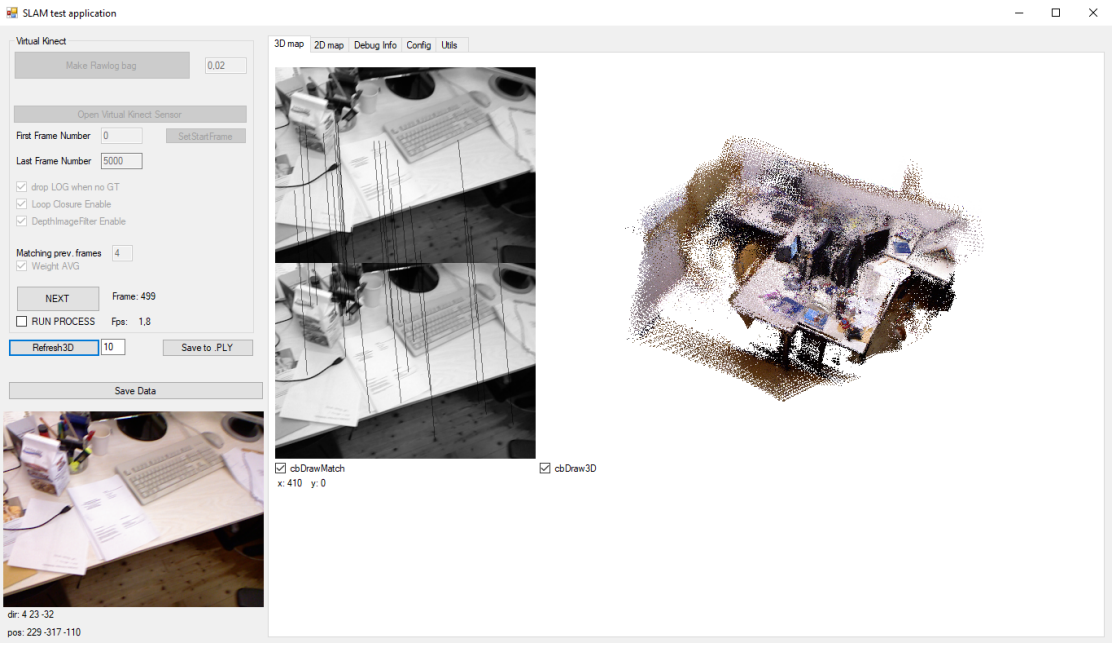
  $$\phi = -2 \cdot \text{atan2}(q_x, q_w) \tag{3.11}$$

  és $\psi = 0$.

3.6. A rendszer működése

Az elkészült tesztalkalmazásom segítségével például a TUM adathalmazokon [63] lehet tesztelni a kifejlesztett eljárásomat, az ISVD algoritmust.

A felhasználói felület (3.6. ábra) bal oldalán találhatók a fő funkciók. A *MakeRawlogbag* gombra kattintva, ki kell választani az adatbázisban található valós pozíciókat tartalmazó



3.6. ábra. SLAM algoritmus teszteléséhez készült rendszer. Az éppen aktuálisan illesztett kép és jellemzőpontjai, valamint az eddig elkészült háromdimenziós pontfelhő látható rajta.

fájlt, majd az alkalmazás egy *mbag* kiterjesztésű állományt készít, amit később meg lehet nyitni az $OpenVirtualKinectSensor$-ra kattintva. Ez a fájl tartalmazza az adatbázisban található szines és mélységi kép párokat, amit például a TUM adatbázis esetén időbélyeg alapján párba kell állítani. A párosítás úgy történik, hogy minden színes képhez megkeresi a rendszer a legközelebbi időbélyeggel rendelkező mélységi képet. A gomb mellett található rubrika, ami alapértelmezetten 0,02s-ra van állítva adja meg, mekkora az az időköz, ami a mélységi és a színes kép időbélyege között már nem elfogadható.

Az $OpenVirtualKinectSensor$-ra kattintva nyitható meg az így elkészített fájl. Alatta a $First$ és $LastFrameNumber$ rubrikákban megadható, hogy hányadik képtől hányadik képig fusson le az algoritmus.

A $DropLOGwhennoGT$ checkboxal be lehet állítani, hogy a végén exportált adatokban maradjanak ki azok a pontok, amik nem rendelkeznek valós pozíció adatokkal, így a végén a benchmark esetén csak azokat a képpárokat vizsgáljuk, amikhez tartozik valós térbeli pozíció.

A *LoopClosureEnable*-lel lehet bekapcsolni a zárt hurkok keresése funkciót. Ez ha be van kapcsolva egy tesztelés során, akkor később a $+LC$ jelzéssel jelölöm.

A *DepthImageFilterEnable*-lel lehet engedélyezni a mélységi képen használható szűrő algoritmusomat (4. algoritmus). Ez ha be van kapcsolva egy tesztelés során, akkor később a $+DIF$ jelzéssel jelölöm.

Ha a fenti két jelölő nincs bekapcsolva, akkor $ISVD$ jelöléssel látom el az eredményeket, mert csak az $ISVD$ algoritmus volt használatban, a mélységi képen nem volt szűrés és nem volt zárt hurok keresés.

A $Matchingprev.Frames$-sel lehet megadni, hogy a jelenleg illesztett képet az ISVD algoritmus hány korábbi képhez próbálja illeszteni. Ezt a későbbiekben MP jelölővel jelölöm, így például 4 kép esetén $MP4$. Valamint a $WeightAVG$ jelölővel lehet megadni, hogy ezek az illesztések sima átlaggal, vagy súlyozott átlaggal adják meg a végső elmozdulást. A tesztek alatt a súlyozott átlagot használtam.

A $Next$ vagy a *Runprocess* gombokkal lehet egyesével, vagy autómatikusan futtatni a programot.

A $Refresh3D$ frissíti a jobb oldalon megjelenő 3D nézetet, a mellette található rubrikában azt lehet megadni, hogy minden n-edik pontot jelenítse csak meg.

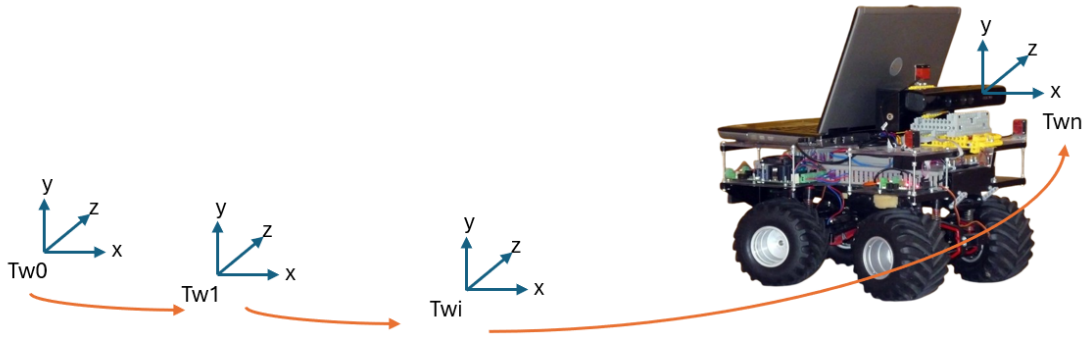
A $Saveto.PLY$ gombbal lehet a 3D rekonstrukciót exportálni fájlba, a későbbi feldolgozáshoz.

Végül a *SaveData* gombbal lehet menteni a becsült elmozdulás adatokat a későbbi utófeldolgozáshoz. Ezek az adatok a *data* könyvtárba kerülnek. Az utófeldolgozásra a *benchmark.m* fájl készül, amit Matlabban lehet futtatni. Ez a fájl megadja az ATE (Absolut Trajektory Error) értéket, valamint kimenetként előállít egy olyan fájlt, amit a TUM benchmark alkalmazással is lehet futtatni.

4 SLAM algoritmus – ISVD

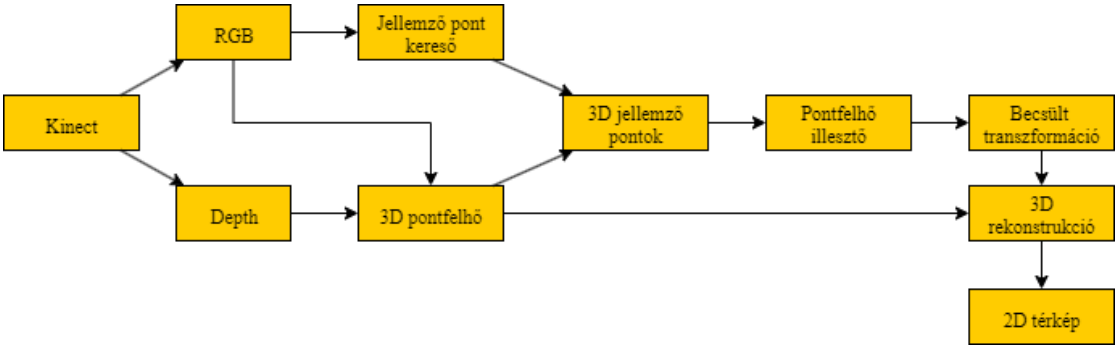
4.1. A rendszer felépítése és fejlődése

A disszertációban bemutatott SLAM rendszer majd egy évtized alatt fokozatosan alakult ki, és érte el mostani működési jellemzőit. A fejlesztés során számos pontosságot javító módszer került a rendszerbe, mint az RGB-D szenzor jelének mélységi szűrése a saját feltételes átlagolóval, az aktuális helyzet becslését előállító transzformáció meghatározásakor az előző szituációk hibával súlyozott beépítése, a jelentős hibát szolgáltató pontpárok intenzívebb szűrése, vagy a zárt hurok detektálásának hatékony megvalósítása, amelyek tovább javították a rendszer működésének minőségi jellemzőit.



4.1. ábra. A jármű mozgása a globális térképen. A $Tw0$ póz jelenti a kiindulási állapotot és a Twn a robot aktuális pózát a térben. Ha a robot elmozdul, a pozíció megváltozik és ha elfordul a robot, az orientáció is változik. Ezt a Twn 4×4 méretű homogén transzformációs mátrix írja le, aminek a balfelső, 3×3 méretű része a forgatómátrixrész, míg az utolsó oszlop az elmozdulás vektor.

A rendszer működésének alapgondolata [29] a következő volt, amit a 4.2 ábra szemléltet. A Kinect szenzorral egymás után készülnek mérések, miközben a robotjármű halad egy munkaterületen. A szenzor aktuális környezetéről készült minden mérés (színes kamerakép (RGB) és pixelszintű mélységi információ (Depth)) egy háromdimenziós pontfelhőt ad.



4.2. ábra. A rendszer első verziójának működését bemutató blokkvázlat

Az új helyen készült pontfelhő illeszthető a korábbi méréshez, ha a közben a szenzor csak kismértékben mozdult el. Az illesztéshez a színes kép jellemzőpontjait (Jellemzőpont kereső) használjuk fel, ahol a mélységi információ is rendelkezésünkre áll. A jellemzőpont keresőnek a SURF eljárást választottam, robosztussága és a viszonylag alacsony futási ideje miatt. Az illesztés során mindig a korábbi jellemzőpontfelhő és az újabb mérés között határozzuk meg az elmozdulást és az elfordulást. Így mindig a legutolsó mérés eredményét adjuk hozzá globális térképhez (4.1. ábra).

A jellemzőpontfelhő illesztő algoritmus már ebben a verzióban is iteratív volt, mindaddig futott a felhők jellemzőpontjainak illesztése, amíg adott hibahatáron belülre nem került a pontosság és az ennél jobban eltérő pontpárok eltávolításra kerültek a számításból. Az egyes ciklusokban adott e_{min} határig feleződött az elvárt pontosság értéke. Az így kapott illesztésnél a jellemzőpontpárok száma körülbelül negyedére csökkent, ugyanakor ezek hibája is sokkal kisebb lett. A módszer segítségével határoztam meg a két felhő közötti tarnszformációt és orientációváltozást, majd pedig az elmozdulás becslését két egymásután készült mérés esetén (5. algoritmus).

A korábbi illesztést és az azelőttieket felhasználva a *Becsült transzformáció* modul segítségével határozom meg a legújabb kamerakép globális pozícióját és orientációját a kiindulási ponthoz képest. Minden új pozícióban készült kép esetén tárolásra kerül egy pozíció és orientáció a rendszer bekapcsolási állapotához képest. A 3D rekonstrukciós modul minden egyes pontfelhőt a korábban meghatározott pozíciójával és orientációjával egymáshoz illeszt, egy nagy globális pontfelhőt kapunk eredményül.

Végül a 2D térkép modul a globális pontfelhőből egy előre meghatározott magasságban kivág egy szeletet, így előállítva egy kétdimenziós térképet.

Az első iteratív SLAM rendszer tehát ICP algoritmus segítségével becsülte meg az

5. algoritmus Többlépéses illesztési eljárás első verzió. A felhők illesztése ICP módszerrel iteratívan.

Input: D, S az aktuális és az azt megelőző jellemzőpontfelhő
Input: $maxIt$ az iterációk maximális száma
Input: e_{start} kezdeti tolerálható hiba
Input: e_{min} elérendő hibanagyság
Input: T_0 kezdeti becslés a transzformációra, vagy ennek hiányában egységmátrix
Output: T az illesztésből becsült orientációváltozás és elmozdulás homogén-koordinátás transzformációs mátrix formájában

1: $T = T_0$
2: $e_n = e_{start}$
3: **for** $j = 0 \rightarrow$ `maxIt` **do**
4: $T = ICP(D, S, T)$ ▷ Klasszikus ICP algoritmus hívása, ahol a D és S halmazok közötti elfordulást és transzlációt T transzformációs mátrix adja vissza
5: **for** $i = 1 \rightarrow |D|$ **do** ▷ $|D|$ az illesztésben szereplő pontpárok száma
6: $S' = T * S$
7: ▷ $s'_i \in S'$ és $d_i \in D$ vizsgálandó pontpárok
8:
9: **if** $\sqrt[2]{(d_{i_x} - s'_{i_x})^2 + (d_{i_y} - s'_{i_y})^2 + (d_{i_z} - s'_{i_z})^2} > e_n$ **then**
10: REMOVE(d_i, s_i)
11: **if** $e_{min} > e_n$ **then**
12: *end algorithm*
13: **else**
14: $e_n = e_n/2$

új pozíciót és orientációt, azonban ez a rendszer az irodalomban megismert hasonló rendszerekhez képest még jelentős hibával rendelkezett, valamint az ICP algoritmus futása sokáig tartott.

A következő változat [30], [31] illesztési algoritmusa annyiban tért el az előzőtől, hogy a jellemzőpontpárok illesztésére SVD módszert használt, $T = ISVD(D, S, T)$ (6. algoritmus). Az $ISVD$ függvény meghatározza a D és S ponthalmazok közötti transzlációs vektorokat és rotációs mátrixokat, ahol jelentős átfedés van a halmazok között. A függvény bemenete a két illesztendő ponthalmaz, illetve ha rendelkezésre áll, akkor egy kezdeti becslés a rotációra és a transzlációra, amennyiben nem, akkor a rotációs rész egységmátrix, a transzláció pedig zérusvektor. A függvény a 2.2. fejezetben leírt módszerrel a ponthalmazok súlypontja alapján állapítja meg, hogy mekkora a transzláció közöttük, majd az SVD felbontással meghatározza a rotációs mátrixot. A T 4×4 transzformáció bal felső sarka a rotációs mátrix lesz, a 4. oszlopa az elmozdulás vektor és homogén koordináták segítségével leírt pontok között teremt kapcsolatot. Az iteráció közben az illesztésben résztvevő jellemzőpontpárok közül eltávolításra kerülnek az aktuális hibaértéknél nagyobb hibát generáló párok. Majd ezt a két lépést ismételgetve egyre pontosabban adja vissza a két halmaz közti transzlációt és rotációt a T transzformációs mátrixban, mivel a tolerálható hiba fokozatosan csökken az iterációk során.

Ez jelentősen felgyorsította a működést és a pontosság is javult (4.1. táblázat). A rendszer működésének minőségét tovább javította, hogy az aktuális pontfelhőt (X_n) nem csak az előző állapothoz (X_{n-1}), hanem néhány azt megelőzőhöz (X_{n-2}), (X_{n-3}), ... is illesztettem (4.3. ábra), és a az illesztések becslését átlagolással határoztam meg [35], amiért a *Becsült transzformáció* modul felel. A megelőző szituációkhoz való hasonlítás lehetővé tette később a zárthurok (LC) kezelésének hatékonyabb megvalósítását is.

A 3D térkép és globális elmozdulás becslés modul minden egyes pontfelhőt a korábban meghatározott pozíciójával és orientációjával egymáshoz illeszt, egy nagy globális pontfelhőt kapunk eredményül.

Végül a 2D térkép modul a globális pontfelhőből egy előre meghatározott magasságban kivág egy szeletet a pontfelhőből, így előállítva egy kétdimenziós térképet.

Az aktuális rendszer folyamatábrája a 4.4. ábrán látható. A rendszer egy RGB-D kamera színes és mélységi képinek frame-to-frame illesztésével megbecsüli a szenzor mozgását. A legtöbb SLAM eljáráshoz hasonlóan ez is két fő részre bontható, egy Front-end és egy Back-end modulra. Az egyes pontfelhők készítése, tárolása, SURF jellemződdetektor futtatása, a továbbfejlesztett 3D jellemzőpont illesztő, kulcskép kereső, frame-to-frame elmozdulás becslés, Loop-closure algoritmus és a sikeres találat esetén a zárt hurok frissítése a Front-end modul része. A Back-end modul csak a 3D rekonstrukciós

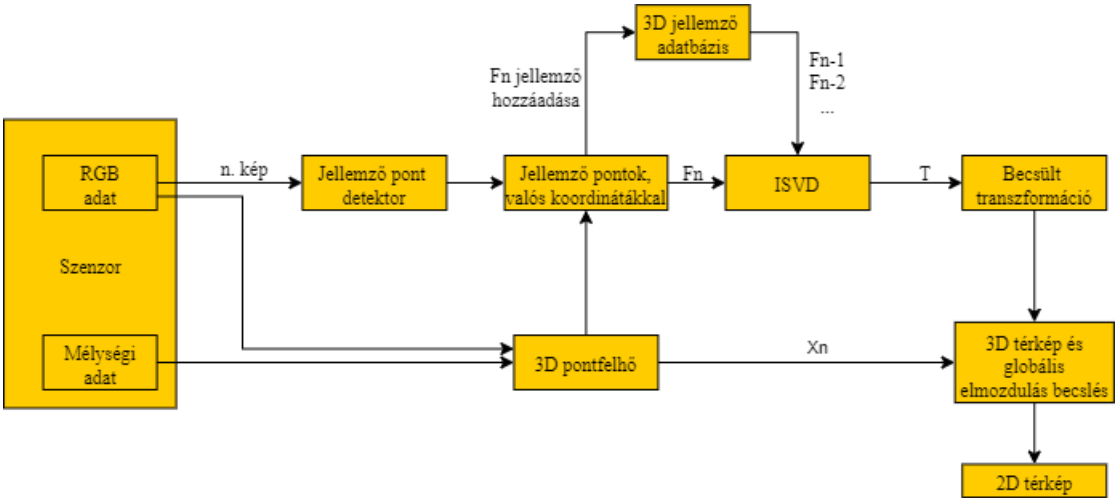
6. algoritmus Pontfelhők illesztése SVD módszerrel iteratívan.

Input: D, S az aktuális és az azt megelőző jellemzőpontfelhő
Input: $maxIt$ az iterációk maximális száma
Input: e_{start} kezdeti tolerálható hiba
Input: e_{min} elérendő hibanagyság
Input: T_0 kezdeti becslés a transzformációra, vagy ennek hiányában egységmátrix
Output: T az illesztésből becsült orientációváltozás és elmozdulás homogén-koordinátás transzformációs mátrix formájában

$T = T_0$
$e_n = e_{start}$
for $j = 0 \rightarrow$ `maxIt` **do**
  $T = SVD(D, S, T)$
  for $i = 1 \rightarrow |D|$ **do** ▷ $|D|$ az illesztésben szereplő jellemzőpontpárok száma
    $S' = T * S$
    ▷ $s'_i \in S'$ és $d_i \in D$ vizsgálandó jellemzőpontpár
    if $\sqrt[2]{(d_{i_x} - s'_{i_x})^2 + (d_{i_y} - s'_{i_y})^2 + (d_{i_z} - s'_{i_z})^2} > e_n$ **then**
      REMOVE(d_i, s_i)
  if $e_{min} > e_n$ **then**
    end algorithm
  else
    $e_n = e_n/2$

4.1. táblázat. Az ICP és az SVD módszerrel meghatározott hiba nagysága egy mérésnél

Módszer	x [mm]	y [mm]	z [mm]	ϕ [rad]	θ [rad]	ψ [rad]	Összhiba [mm]	Összhiba [rad]
ICP	-183,3	617,4	-231,1	-0,0365	0,0019	-0,1124	343,8	0,05
SVD	-99,2	380,3	-264,7	-0,0336	0,0058	-0,0404	**248**	**0,026**



4.3. ábra. A rendszer harmadik verziójának blokkdiagramja

eljárást tartalmazza, mivel a front-end részben minden új pozíció becslése és korrekciója megtörténik. A végső 3D modell generálása nem on-line történik, de az útvonal folyamatos frissítése on-line. A működés részleteit a következő alfejezetek mutatják be.

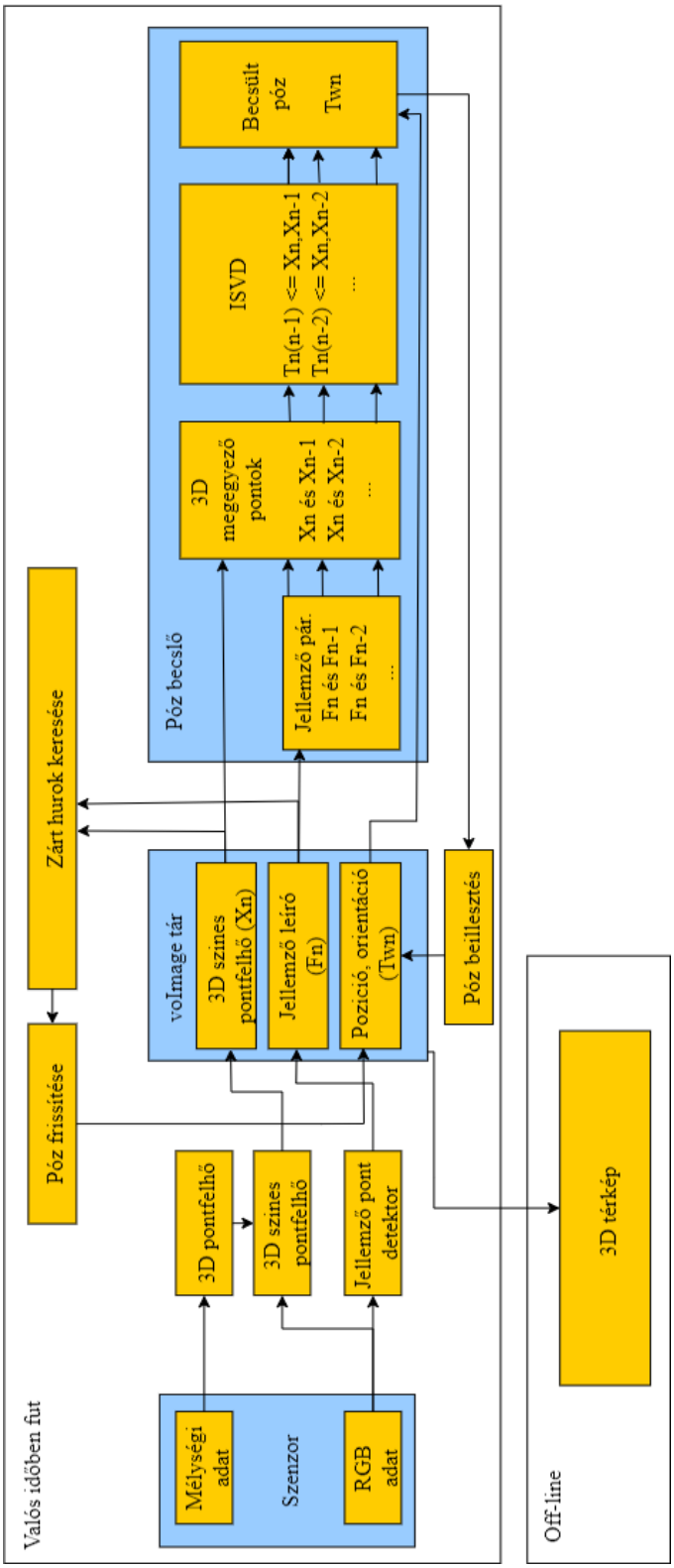
4.1.1. 3D pontfelhő indexelése

A rendszer sorra dolgozza fel az RGB-D kamera színes ($IRGB_n$) és mélységi (ID_n) képeit, ahol n a feldolgozott kép sorszámát jelöli. A legutolsó mélységi adatból, 3D pontfelhőt generál, amihez hozzárendeli a színes kameraképen lévő színeket is, ez lesz a 3D színes pontfelhő (X_n). A pontfelhő minden eleme a kamera lokális koordináta-rendszerében tartalmazza a háromdimenziós térbeli pont koordinátáit és a hozzá tartozó szín értékét.

Az aktuális RGB-D kameraképből elő kell állítani (SURF jellemződetektor segítségével) egy pontfelhőt (jellemzőpontok felhője), ehhez a színes képen jellemzőpontokat kell keresni, végül ezeket az adatokat tárolni kell a későbbi felhasználásra. Ezért az RGB-D kamera színes adataiból jellemzőpont leírók halmazát (F_n) generálja a rendszer. A pontfelhő minden pontja a színes kamerakép pixel pozíciójával kerül indexelésre, a későbbi 3D jellemzőpontok gyors megfeleltetése miatt.

4.1.2. Jellemzőpont leírók

Az egymásutáni képek közti elmozdulás becsléshez szükséges, hogy a hozzá tartozó képek és pontfelhők részben átfedésben legyenek egymással. A pontfelhők közös részéből a



4.4. ábra. SLAM rendszer aktuális felépítése

rendszer kiválaszt néhány olyan pontot, ami mind a két pontfelhőben megtalálható. A kiválasztás a színes kameraképen jellemzőleírók segítségével történik. Minden új frame esetében, egyszer a színes kameraképen SURF jellemzőpontok keresése történik, majd a jellemzőleíró vektor (F_n) eltárolásra kerül a korábban elkészült 3D pontfelhővel együtt a *voImage* tárba. A SURF detektor küszöbértéke, a képek élessége alapján változtatásra kerül, ha egy kép túl kevés jellemzőpontot tartalmazna a küszöbérték alacsonyabb értékűre csökken, ezzel biztosítva automatikus igazodás a környezeti feltételek változásához.

Korábbi vizsgálatok során kipróbáltam különböző jellemződetektorokat, így a SIFT, SURF, ORB, FAST jellemződetektorokat[64], [65], [66], [67] is. A választásom egy robosztusabb jellemződetektorra esett, ami a transzformációkra robosztusabb eredményt szolgáltat. Így a SIFT és SURF detektor közül a SURF detektort a sebessége [68] miatt választottam, amit az EMGUCV [69] könyvtáron keresztül használok.

A rendszer folyamatosan kulcsképkockákat választ ki az új színes kameraképekből. A Loop-closure algoritmus folyamatosan figyeli az új kép és a kulcsképkockák közti egyezést, a jellemzőleírók segítségével. Ha hasonló helyzetbe került a szenzor, mint korábban volt, akkor pontosítható a jelenlegi becsült helyzet. A kulcsképkockák kiválasztása két szempont szerint történik. 20 képenként mindenképpen kiválasztásra kerül egy kulcskép, úgy, hogy az elmúlt 20 kép közül a legélesebb, legtöbb jellemzőpontot tartalmazó kép lesz az. Az élesség meghatározására a jellemzőleírókat használja fel a rendszer.

A *voImage* tár csak egy része van folyamatosan jelen a memóriában, így például a jellemzőleíró (F_n) és a becsült póz adatok a kiindulási koordináta-rendszerben, minden egyes pontfelhőhöz (Tw) folyamatosan rendelkezésre állnak a memóriában a gyors elérés céljából. Ezeknek az adatoknak a kis mérete lehetőséget biztosít, hogy a memóriában tartsuk azokat folyamatosan. Mindig csak a 3D színes pontfelhő (X_n) legújabb példánya van betöltve a memóriába, a nagy mérete miatt, illetve a Loop Closure illesztés idején van még rá szükség a Front-end modulban. A régebbi adatok ideiglenesen archiválásra kerülnek a háttértárra. A gyors mentés miatt a tesztek alatt erre a feladatra SSD-t használtam. A Loop-closure algoritmus a jellemzőleírók közötti egyezést vizsgálva, próbálja meghatározni, hogy az aktuális kamerakép megegyezik-e részben egy korábbi kulcsképpel. Egyezés esetén visszatöltésre kerül a 3D pontfelhő, hogy az éppen egyező jellemzőleírókhoz meg lehessen határozni a szenzor lokális koordináta-rendszerében a térbeli helyzetét. Így a nagy memóriaigényű adatok csak rövid időre foglalják a memóriát. A rendszer nagy mennyiségű adattal is működik, viszonylag alacsony (körülbelül 2 GB) memória használat mellett.

4.2. Továbbfejlesztett pozícióbecslő eljárás, ISVD algoritmus

Az új kamerakép pozícióbecslése az előre előállított adatok (*voImage* tár) segítségével történik. Az új pozíció becslése a továbbfejlesztett többlépéses illesztési eljárással ($ISVD$) valósul meg (7. algoritmus), ami az SVD felbontáson alapul. A rendszer mindig a legutolsó, n-edik képet próbálja illeszti, az előzőleg becsült legutóbbi (MP) darab kamerapozíciókhoz az $(n-1)$, $(n-2)$,...,$(n-MP)$ korábbi tárolt adatok segítségével $(X_{n-1}, F_{n-1}), (X_{n-2}, F_{n-2}), ..., (X_{n-MP}, F_{n-MP}))$, végül az új becsült transzformáció (Tw_n) elmozdulás része különböző súlyozásokkal kerül véglegesítésre.

A rendszer indulásakor az első kamerakép pozíciója a globális koordináta-rendszer origója lesz. Az új kamerakép mindig a közvetlen előtte levő néhány kameraképhez kerül illesztésre. A rendszerben az adott kép néhány (MP darab) korábbi képhez van illesztve, a tesztelések szerint 4-nél kisebb számú illesztéssel pontatlanabb térkép állítható elő. Ennél több korábbi képhez történő illesztéssel azonban már csak kis mértékben javul a pozíció becslés, de a futási idő jelentősen megnövekszik (lásd majd 5.2. táblázat). Az adott kép pozícióbecslése az előző néhány képhez párhuzamosan futtatva, több szálon történik. A rendszer többi része nem párhuzamosított, csak a többlépéses illesztési folyamat fut külön szálakon.

Az aktuális frame a korábbi frame-hez történő illesztés első lépése során kiválasztunk egy redukált ponthalmazt az aktuális és egy megelőző színes kameraképből. A függvényhívások $match(X_n, F_n, X_{n-1}, F_{n-1})$, $match(X_n, F_n, X_{n-2}, F_{n-2})$, ..., jellemzőpontok alapján meghatározzák azokat a színes kamerakép pixeleket, amik megegyeznek a két képen (a 8. algoritmus bemutatja két egymásutáni szineskép jellemzőpontjainak párosítását, illetve a módszer magyarázata is később bemutatásra kerül), majd a korábban előállított indexelt pontfelhőből az egyező pontokhoz hozzárendelik a térbeli koordinátájukat. Így két redukált pontfelhőt kapunk, amiben minden pontnak megvan a hozzátartozó térbeli párja (D_{n-f}, S_{n-f}).

A következő lépésben meghatározásra kerül a két szomszédos kép között a kamera relatív elmozdulása. A kiinduló $Tw_{n-f} = T_0$ transzformációt a D_{n-f} és S_{n-f} halmaz között, a rendszer több lépés során pontosítja. Minden iteráció során SVD felbontással meghatározásra kerül a két ponthalmaz közti transzformáció (Tw_{n-f}), majd ezt felhasználva próbaillesztést hajt végre. A próbaillesztés során a rendszer S halmazt a kapott transzformáció segítségével a D halmaz koordináta-rendszerébe konvertálja $(S' = Tw_{n-f} * S_{n-f})$. Végül megnézi az illesztés pontosságát, valamint összetartozó pontpárok euklideszi távolságát a hibás pontpárok törlése miatt (4.5 ábra).

Az illesztés minőségének megállapítására (4.1) a próbaillesztés után a pontpárok átlagos

7. algoritmus Többlépéses illesztési eljárás (ISLAM) pontfelhők közötti transzformáció meghatározására.

Input: MP az illesztésben résztvevő előző pontfelhők száma
Input: $X_n, X_{n-1}, ..., X_{n-MP}$ az $n.$ és az azt megelőző szinezett pontfelhők
Input: $F_n, F_{n-1}, ..., F_{n-MP}$ jellemzőpontok halmaza az $n.$ és az azt megelőző képeken
Input: $maxIt$ az iterációk maximális száma
Input: $minError$ a hiba legkisebb értéke
Input: $minPoint$ a pontpárok minimális száma
Input: α, β az akceptálható hibát meghatározó konstansok
Input: T_0 kezdeti becslés a transzformációra, vagy ennek hiányában egységmátrix
Output: Tw_n az egyes illesztésekből súlyozottan becsült elmozdulást tartalmazó transzformáció
Output: W_{n-f} az egyes illesztések súlya

for $f = 1 \rightarrow MP$ **do** ▷ Korábbi állapotokhoz illesztés külön szálakon
  $D_{n-f}, S_{n-f} = match(X_n, F_n, X_{n-f}, F_{n-f})$ ▷ A színes kamerakép jellemző leíró vektorait felhasználva megadja a két kép összetartozó pontpárjait és ebből meghatározza a 3D összetartozó pontpárokat. Lásd 8. algoritmus
  $Tw_{n-f} = T_0$
  for $j = 1 \rightarrow maxIt$ **do**
    $Tw_{n-f} = SVD(D_{n-f}, S_{n-f}, Tw_{n-f})$
    $th = \alpha * (j)^{\beta}$

    $S' = Tw_{n-f} * S_{n-f}$
    $D = D_{n-f}$
    $e = 0;$
    ▷ $s'_i \in S'$ és $d_i \in D$ vizsgálandó pontpárok
    for $i = 1 \rightarrow |D|$ **do** ▷ $|D|$ az illesztésben szereplő pontpárok száma
      $e = e + |d_{i_x} - s'_{i_x}| + |d_{i_y} - s'_{i_y}| + |d_{i_z} - s'_{i_z}|$
    $W_{n-f} = 1/(e/|D|)$

    for $i = 1 \rightarrow |D|$ **do**
      if $\sqrt[2]{(d_{i_x} - s'_{i_x})^2 + (d_{i_y} - s'_{i_y})^2 + (d_{i_z} - s'_{i_z})^2} > th$ **then**
        $remove(d_i, s_i)$
    if $th < minError$ OR $|D| < minPoint$ **then**
      end iteration
        ▷ A szálak bevárása és utána az elmozdulások súlyozott kiértékelése
$t = 0$
$w = 0$
for $f = 1 \rightarrow MP$ **do**
  $t_{n-f} = Tw_{n-f}$ mátrix 4. oszlopa ▷ az elmozdulás vektor
  $t = t + t_{n-f} * W_{n-f}$
  $w = w + W_{n-f}$
Tw_n 4. oszlopa $= t/w$

távolságát veszi alapul a rendszer, majd ennek a reciproka adja az illesztés minőségét jelentő súlyt (W_{n-f}).

$$W_{n-f} = \frac{1}{\frac{1}{|D|} * \sum_{i=0}^{|D|}(|d_{i_x} - s'_{i_x}| + |d_{i_y} - s'_{i_y}| + |d_{i_z} - s'_{i_z}|)} \tag{4.1}$$

Ez az érték minél nagyobb, annál közelebb kerültek az illesztés után a pontok, így kevesebb hibás pontpárt tartalmaz a redukált halmaz, valamint kisebb a ponthalmazok zaj értéke. A hibásan detektált pontpárok távolsága nagyobb lesz, mint a többié, ezért minden lépésben töröljük azokat, amiknek a távolsága egy th küszöbértéknél nagyobb. A küszöbérték egy hatvány függvény szerint csökken az iterációk során (4.2), ahol j az illesztési iteráció sorszáma. A tesztelések során a függvény paramétereit a következőkre választottam: $\alpha = 1200$ és $\beta = -2,6$.

$$th = \alpha * (j)^{\beta} \tag{4.2}$$

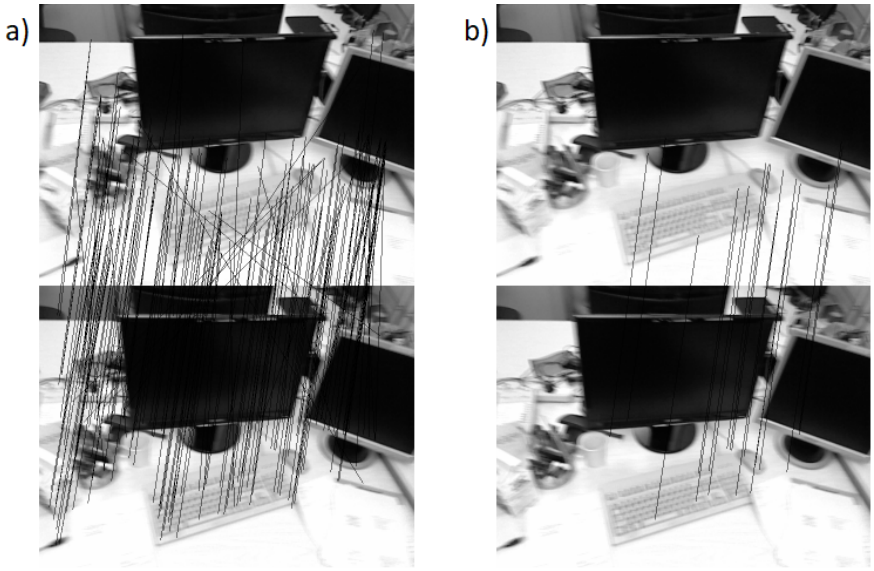
Az egyes illesztési becslés akkor ér véget, ha a ciklus elért egy maximális iteráció számot ($maxIt$), vagy túl kevés pont maradna a nem összetartozó pontpárok törlése után ($minPoint$). Bevárva a párhuzamos szálakat, a következő részben a transzformáció elmozdulás értékét súlyozva pontosítjuk az aktuális kép és a korábbi képek illesztési eredményeinek elmozdulás részét. A korábbi képek térbeli helyzetéhez hozzáadva a súlyozott elmozdulást, megkapjuk az adott kép globális térbeli helyzetét a korábbi néhány kép szemszögéből. Tehát a transzformációk homogén koordintás leírásából az utolsó, elmozdulást reprezentáló, 4. oszlopot kiemeljük, ezeket az értékeket a korábban kiszámolt súlyokkal vesszük figyelembe, és így előáll az aktuális kép térbeli pozíciójának pontosabb helyzete a transzformáció utolsó oszlopába beírva (Tw_n).

A 8. algoritmus bemutatja a *match* függvényt, amely a RANSAC módszert használja a homográfia transzformáció számításához, és visszaadja a legjobb homográfia segítségével meghatározott (inlier) pontpárokat, elvégezve a 2.3 fejezetben megadott regisztrációs műveletett a két kép jellemzőpontjai között. Az algoritmus bemutatása a következő:

Inicializálás: Meghatározzuk a maximális iterációk számát (N), inicializáljuk a legjobb homográfia mátrixot H_{best} és a legnagyobb helyesen párosítottak (un.inlier-ek) számát $max\_inliers$. Emellett itt inicializáljuk a legjobb inlier pontpárokat is D_1, S_1.

Véletlen mintavétel: Minden iterációban véletlenszerűen kiválasztásra kerül 4 pontpár az első és második képből, hogy kiszámítsuk a homográfiát.

Homográfia számítása: A kiválasztott 4 pontpár alapján homográfia trfanszformáció meghatározása.



4.5. ábra. ISVD algoritmus által kiszűrt nem összetartozó és pontatlan jellemzőpárok. Az a) képen a kezdeti állapot látható, a SURF jellemződetektor által meghatározott jellemzőpárokkal, a b) képen az ISVD algoritmus 25. iterációja után megmaradt jellemzőpárok láthatóak.

Inlier-ek meghatározása: Minden pontpárra alkalmazzuk a homográfiát, és kiszámítom, hogy a transzformált pont és a megfelelő pont közötti távolság kisebb-e egy adott küszöbnél (ϵ). Azokat a pontpárokat, amelyek megfelelnek a feltételnek, inlier-nek tekinthetők, és hozzáadjuk az aktuális iteráció inlier pontpárjaihoz.

Legjobb homográfia frissítése: Ha az aktuális iteráció több inlier-t tartalmaz, mint az eddigi legjobb iteráció, akkor frissítjük a legjobb homográfia transzformációmátrixot, és eltároljuk az új legjobb inlier pontpárokat is.

Visszatérés: Az algoritmus végén a legjobb homográfia transzformációhoz tartozó inlier pontpárokat visszaadjuk.

Az algoritmus megvalósításánál felhasználtam a korábban már említett EMGUCV könyvtárat.

8. algoritmus RANSAC-alapú homográfia számítás segítségével az egymásnak megfeletett (inlier) pontpárok számítása, a 7. algoritmus *match* függvénye

Input: X_1, X_2 ▷ Bemeneti 3D színes képek
Input: F_1, F_2 ▷ Bemeneti színes képek jellemzőpontjai, pl. SURF jellemzőkkel meghatározva
Output: D_1,S_1 InlierPontPárok ▷ A legjobb homográfia transzformáció által meghatározott (inlier) pontpárok

Inicializálás
$N \leftarrow$ max iter ▷ Maximális iterációk száma
$H_{\text{best}} \leftarrow \emptyset$ ▷ Legjobb homográfia inicializálása
max_inliers $\leftarrow 0$ ▷ Maximális inlier-ek száma
D_1,$S_1 \leftarrow \emptyset$ ▷ Legjobb inlier pontpárok inicializálása
for $i = 1$ to N **do**
    Véletlen mintavétel
    $S \leftarrow$ véletlen 4 pontpár (F_1, F_2) jellemzőpontokból
    Homográfia számítása
    $H \leftarrow$ Homográfia számítása a 4 pontpárból S használatával
    Inlier-ek meghatározása
    inliers $\leftarrow 0$
    InlierPontPárok $\leftarrow \emptyset$ ▷ Aktuális iteráció inlier pontpárjai
    for minden $(p_1, p_2) \in (F_1, F_2)$ **do**
        $p_2' \leftarrow H \cdot p_1$ ▷ Alkalmazd a homográfiát az első kép pontjára
        if $\|p_2' - p_2\| < \epsilon$ **then** ▷ Az inlier feltétel ellenőrzése
            inliers $\leftarrow$ inliers $+ 1$
            InlierPontPárok $\leftarrow$ InlierPontPárok $\cup \{(p_1, p_2)\}$ ▷ Pontpár hozzáadása az inlierekhez
    Legjobb homográfia frissítése
    if inliers $>$ max_inliers **then**
        $H_{\text{best}} \leftarrow H$
        max_inliers $\leftarrow$ inliers
        $D_1, S_1 \leftarrow$ InlierPontPárok ▷ Frissítsd a legjobb inlier pontpárokat
return D_1, S_1

4.3. Zárt hurok (Loop-closure) detektálás

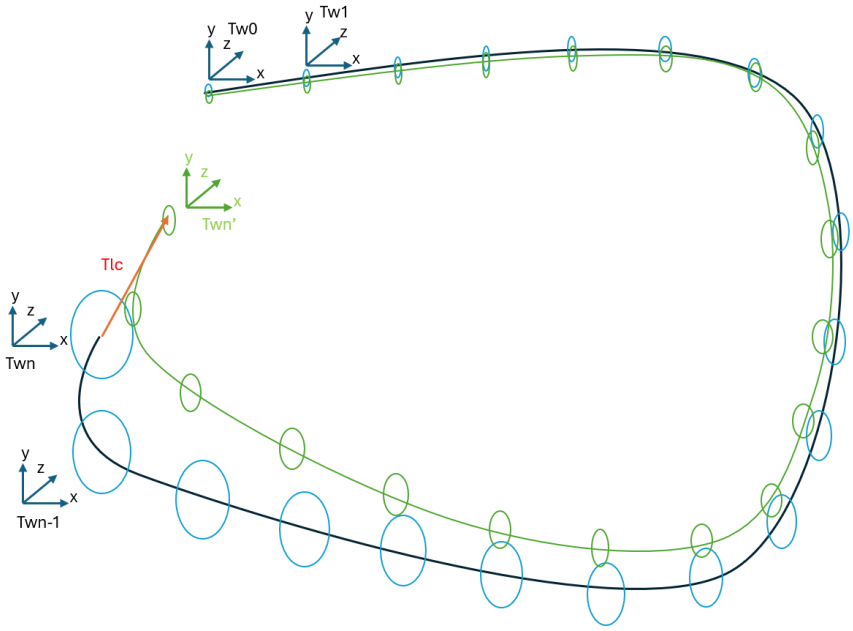
A Front-end modul utolsó lépésként a Loop-closure (LC) algoritmussal folyamatosan figyeli az aktuális kép és a kulcsképkockák közti egyezéseket. Az adott kép jellemzőleíróit a korábbi kulcsképek jellemzőleíróival összehasonlítva határozza meg az egyezést. Ha egyezést talál egy korábbi képpel, akkor a többlépéses illesztéssel (7. algoritmus) megpróbál relatív elmozdulást és orientációt (Tlc) meghatározni a kulcskép és az aktuális kép között. A rendszer nem vizsgál meg minden aktuális képet, csak akkor fut le a Loop-closure algoritmus, ha az adott kép megfelelő élességgel rendelkezik. Az élesség megállapítása a korábbi 20 kép jellemzőleíróinak alapján történik. Ha a legutóbbi képeken talált jellemzőleíró pontok számának a mediánértékénél 1,2-szer több az aktuális kép jellemzőleírók száma, akkor ez a kép kevésbé elmosódott és részletgazdagabb, mint a többi.

Ezután a rendszer megvizsgálja, hogy az aktuális kép pozíciója és a kulcskép globális pozíciója közel helyezkedik-e egymáshoz, azaz a szenzor hasonló irányba néz-e és hasonló pozícióban áll-e, mint korábban. Ha nagy az eltérés, akkor nem vizsgálja meg a kulcsképet, mert vagy hamisan detektált esetről van szó, vagy nagyon kis átfedés lehet a kulcskép és az aktuális kép között.

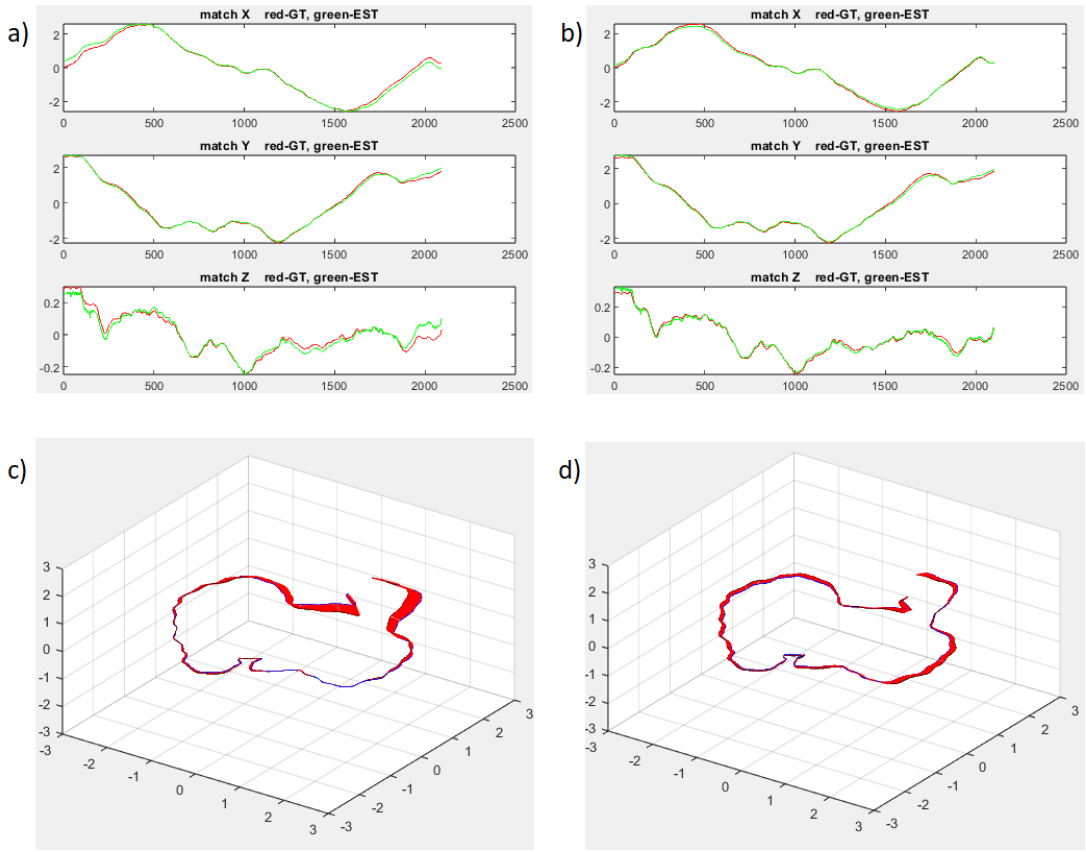
Amennyiben talál a rendszer egy zárt hurkot, akkor az aktuális, és a korábbi pózokat korrigálja (4.6. ábra). A jelenlegi becsült póz és a LC algoritmus által talált új póz közötti transzformációt a Tlc mátrix tartalmazza. Az aktuális pózt a Tlc eltéréssel a korábbi pózokat ennek arányosan kisebb eltéréseivel korrigálja a rendszer.

Így az algoritmus futási ideje jelentősen csökken, mert kisszámú kulcsképet kell megvizsgálni a többlépéses illesztéssel, aminek nem elhanyagolható a futási ideje a jellemzőpárok keresése, illetve a többlépéses illesztés miatt.

A 4.6. ábrán látható az $FB3/long\_office\_household$ adatsoron történt vizsgálat az LC alagoritmussal. Itt az ISVD algoritmus MP=4 és DIF szűrő mellett a 2100. képkockáig futott, ahol az első nagy zárt hurok található ebben az adathalmazban. Az ábrán látható hogy az LC algoritmust futtatva az abszolút hiba nagysága kisebb, mintha nem használtuk volna az algoritmust. A táblázatban (4.2. táblázat) látható az ATE RMSE érték méterben megadva a 2100 kép esetén futtatva az ISVD+DIF és az ISVD+DIF+LC eljárást. Látható hogy az LC algoritmus esetén jelentősen kissebb az abszolút hiba (ATE) értéke (4.7. ábra).



4.6. ábra. Zárt hurok detektálása. Feketével a becsült útvonal, pirossal a detektált eltérés nagysága és zöld színnel a LC algoritmus futtatása utáni utvonal. Az n. állapotban sikerült detektálni azt a részt, amit a szenzor a 0. állapotban látott, így pontosítható a megtett út minden közbenső helyzete. Az ellipszisek az akkumulálódott hibák nagyságát szimbolizálják.



4.7. ábra. Loop Closure algoritmus eredménye. Az $FB3/long\_office\_household$ adathalmazon ISVD algoritmus MP=4 és DIF szűrő mellett a 2100. képkockáig, ahol az első nagy zárt hurok itt található. Az a) képen a LC előtti, a b) képen az LC utáni eredmény látható, piros színnel a valós, zöld színnel a becsült x,y és z koordináta irányban a történt elmozdulás (méterben) a képkockák függvényében (vízszintes tengely). Valamint a c) képen a LC előtti, a d) képen az LC utáni eredmény látható, feketével a valós, kékkel a becsült útvonal és pirossal az abszolút hiba látható. Az a) és b) képen az y tengelyen méterben, az x tengelyen a képkocka sorszáma látható. A c) és d) képen a tengelyek méterben vannak megadva.

4.2. táblázat. Az ISVD+DIF és az ISVD+DIF+LC eljárás futtatva az *FB3/long_office_household* adatlahmazon. Az ATE RMSE értékek méterben megadva a 2100 kép feldolgozása után.

Adatbázis	ISVD+DIF	ISVD+DIF+LC
$FB3/long\_office\_household$	0.1809	0.1094

5 Benchmark

5.1. Adathalmazok és tesztkörnyezet

A bemutatott SLAM rendszer a releváns, hasonló SLAM eljárásokkal összehasonlításra került. A tesztek minden esetben a következő konfigurációjú számítógépen futottak: AMD Ryzen 5 5600x, 6 magos processzor és 32GB RAM. Az összehasonlítás során az algoritmus paraméterbeállításai azonosak voltak minden adathalmaz esetében. A rendszer pontosságának vizsgálata két különböző forrásból származó adatsoron történt. Az egyik nagy online elérhető adathalmaz a Technical University of Munich több különböző Kinect szenzorral készült off-line adathalmaza [63]. A teszteket az fr1, fr2 és fr3 csoportba tartozó adathalmazaikon futtattam. Második forrásként a Ponznani Egyetemen készült adathalmaz került felhasználásra [34]. Ezt mobil robot fejlesztésekhez készítették és tették elérhetővé. Az eredeti 4 darab mérési sort kiegészítették több új adathalmazzal, ahol Kinect 1 és 2 szenzorokkal is megismételték a rögzítést. Mind a két adathalmaz tartalmaz minden egyes pozícióban készült képhez valós térbeli pozíciót (bár néhány adathalmaznál előfordulnak kimaradások a valós térbeli pozíció esetén). Ezenkívül saját adathalmazokon is futtattam az eljárásomat, azonban ezek az adathalmazok nem tartalmaztak abszolút kamerapozíciókat, így itt csak a teljes akkumulálódott hibát lehetett vizsgálni.

A következő fejezetekben először bemutatásra kerül, hogy a szakirodalomban milyen mérőszámokkal szokták vizsgálni a SLAM rendszerek pontosságát, majd pedig a következő részben azt vizsgálom, hogy a az algoritmusom pontossága miként változik az iterációk függvényében. A minőséget az is meghatározza, hogy az aktuális pontfelhőt (képet) hány megelőző pontfelhőhöz hasonlítjuk, hiszen ezek eredményei is beépülnek a meghatározott hibanagyságukkal súlyozva az aktuális eredménybe. A saját adathalmaz elkészítéséről és a kapott eredményekről szól a következő alfejezet, majd pedig a Poznani Egyetem adatsorán vizsgáltam a saját módszerem pontosságát. Végül a Müncheni Műszaki egyetem adathalmazait felhasználva hasonlítottam össze saját módszerem eredményeit a szakirodalomban megtalálható más módszerekével.

5.2. Összehasonlításhoz használt mérőszámok

A legtöbb hasonló kutatásban a becsült elmozdulás pontosságának a megállapítására gyakran használják az Absolute Trajectory Error (ATE) mérőszámot [70]. Az ATE érték megadja, hogy a teljes becsült útvonal és a valós pozíció között mekkora eltérés tapasztalható. Az elmozdulás becslés esetén, az egyes illesztésekkor keletkezett kis hibák összeadódnak, ezért ha hiba keletkezett egy illesztésnél, az már a többin is megjelenik. Az ATE értékeket a becsült és a valós út eltérésére méterben vizsgáltam. Az alábbi táblázatokban az ATE értékek Root Mean Square Error összegét hasonlítottam össze hasonló munkák eredményeivel, valamint a saját algoritmusom különböző paraméterei esetén is vizsgáltam az ATE értéket.

A Relative Pose Error (RPE) mérőszám két egymásután készült kép közötti relatív elmozdulást vizsgálja csak. Az elmozdulásbecslés esetén, az egyes illesztésekkor keletkezett kis hibák összeadódnak, így ha hiba keletkezett egy illesztésnél, az már a többin is megjelenik. A RPE az aktuális illesztés hibáját veszi alapul. A jelenlegi és az azt megelőző valós és becsült elmozdulás különbségéből számolható [33].

A saját adathalmazok vizsgálatánál a következő módon vizsgáltam a pontosságot. A kezdő pozíció és az utolsó pozicó hasonló helyzetben készült képeket tartalmazott. Az útvonal bejárása után a teljes akkumulálódott hibát vizsgáltam.

5.3. ISVD algoritmus pontossága az iterációk függvényében

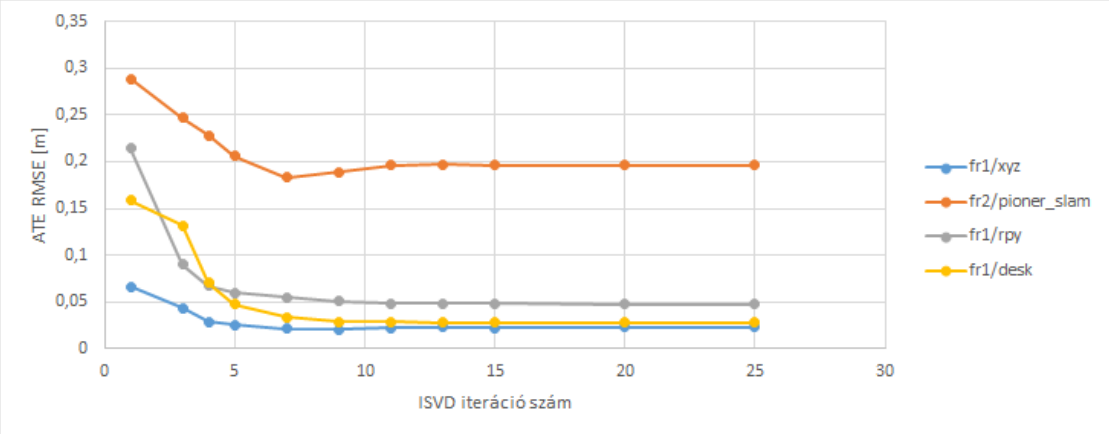
Az alábbi részben az ISVD algoritmus pontosságát vizsgáltam az alkalmazott iteráció szám függvényében. Az algoritmus (7. algoritmus) minden iterációban megbecsüli a két ponhalmaz közötti tarnszformációt, majd egy egyre csökkenő érték szerint törli azokat a pontpárokat, amik nem illenek bele az illesztésbe. Minden iteráció során egyre kevesebb nem a pontfelhőbe illő pontpár marad meg.

Az 5.1. táblázatban és az eredményeket megjelenítve grafikonon (5.1. ábra) 25 iteráció során vizsgáltam algoritmusom pontosság (ATE RMSE) értékét a Müncheni Egyetem 4 adathalmazán. Minden esetben igaz, hogy az iterációk kezdetén jelentős a javulás a nagy hibát generáló pontpárok fokozott elhagyásával. Néhány adathalmaznál azonban egy idő után a hibák minimálisan növekednek. Ennek oka, hogy egyre kevesebb pontpár marad meg, amiknek az illesztése kismértékben ront a pontosságon.

A táblázat utolsó sorában az ”Auto” bejegyzés azt jelenti, hogy az iterációk számát két küszöbérték határozza meg, amit egy minimális illesztési hiba vagy egy minimális megmaradó pontpár ad meg.

5.1. táblázat. ISVD algoritmus pontossága (ATE értékek) az iterációk számának függvényében. A táblázatban látható az ISVD algoritmus 1.-től a 25. iterációig 4 adathalmaz esetében az ATE érték változása, ami a kezdeti iterációk esetében nagy mértékben csökken.

Iterációk száma	fr1/xyz	fr2/pioner_slam	fr1/rpy	fr1/desk
1	0,0659	0,2887	0,2138	0,1589
3	0,0435	0,247	0,0896	0,1319
4	0,0284	0,2284	0,0674	0,07
5	0,0252	0,2061	0,0596	0,0478
7	0,0212	0,183	0,0551	0,0337
9	0,0206	0,1892	0,0506	0,0288
11	0,0218	0,196	0,0479	0,0284
13	0,0228	0,1972	0,0482	0,0277
15	0,0224	0,1966	0,048	0,0277
20	0,0228	0,196	0,0472	0,0277
25	0,0226	0,196	0,0472	0,0277
Auto	0,0204	0,1829	0,04999	0,0302



5.1. ábra. ATE érték az ISVD eljárás iterációinak függvényében. A grafikonon látható az ISVD algoritmus 1.-től a 25. iterációig 4 adathalmaz esetében az ATE érték változása, ami a kezdeti iterációk esetében nagy mértékben csökken.

5.2. táblázat. ISVD algoritmus pontossága a megelőző képek számának függvényeként. A teszt során nem volt LC és DIF szűrő bekapcsolva, csak az ISVD algoritmus eredményei láthatóak a táblázatban.

Teszthalmaz	MP:1	MP:2	MP:3	MP:4	MP:6
fr1/rpy	0,0527	0,0493	0,0474	**0,0466**	0,0472
fr1/xyz	0,0216	0,0198	**0,0197**	0,0232	0,0236
fr1/360	0,1406	0,1235	0,1102	0,1028	**0,0864**
fr1/desk	0,0878	0,0663	0,0476	0,0455	**0,0422**
fr1/desk2	0,0882	0,0964	0,089	**0,0847**	0,0963
fr1/plant	0,0747	0,0646	0,0581	0,0517	**0,0469**
fr2/desk	0,6204	0,4815	0,4042	0,3431	**0,2582**
fr2/pioner 360	0,8295	0,6736	0,597	0,5301	**0,4643**
fr3/walking xyz	0,6422	0,4608	0,335	0,2696	**0,1781**
Átlag	0,27827	0,22852	0,19788	0,177	0,15212

5.4. ISVD algoritmus pontossága a megelőző illesztendő képek számával összevetve

Az 5.2. táblázat a Müncheni Egyetem különböző teszthalmazain (sorok) mutatja be a pontosság változását annak függvényében, hogy hány megelőző képpel (*MP*) lett összehasonlítva az aktuális, és milyen javulást eredményezett a korábbi összehasonlítások beépítése az aktuális eredmény pontosságára. A tapasztalatok szerint az átlagos pontosság javul ha több megelőző képpel hasonlítjuk össze az aktuálist, de előfordulhat olyan eset, amikor az adott adathalmaznál (pl. *fr1/xyz*) nincs folyamatos javulás. Azonban ilyenkor is igaz, hogy a 6 előző szituációval történő összehasonlítás eredményét beépítve az aktuális elmozdulásba, minimálisan nagyobb a hiba a legjobb értékhez képest.

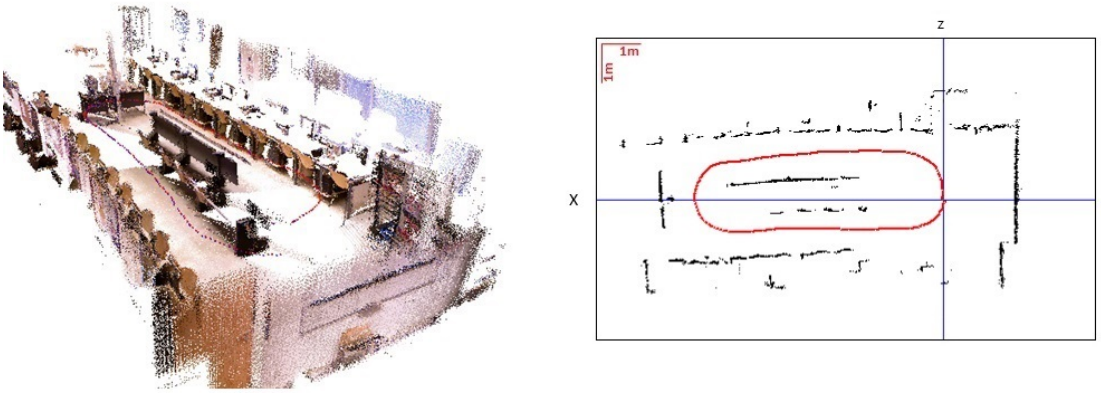
5.5. Tesztelés saját adathalmazon

Az eredmények összehasonlíthatósága érdekében az algoritmus tesztelése három különböző forrásból származó adatsoron történt. Az egyetem laborjaiban és folyosóin is készültek adathalmazok a saját robotrendszerrel. Valamint második és harmadik forrásként a Ponznani Egyetemen készült adathalmaz és a TUM adathalmazok kerültek felhasználásra. Először az egyetemen készült adatsorok tesztelése kerül bemutatásra.

A saját mérések esetében nem áll rendelkezésre a szenzor valós (abszolút) térbeli

5.3. táblázat. Összes akkumulálódott transzlációs (milliméterben) és rotációs (fokban) hiba. Az y tengelyen a magasság értéke szerepel, és e körül fordult el a robot (ψ), x és z a mozgás síkja.

Adatsor	x	y	z	ϕ	θ	ψ
labor213	-250	2,5	15	0,9	-1	9

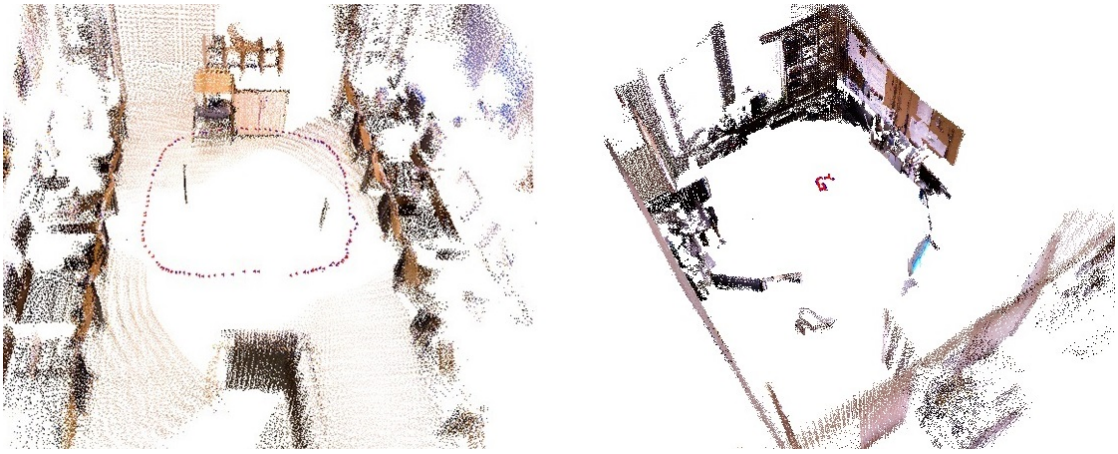


5.2. ábra. Az egyik saját adatsoron futtatott eredmény. Az egyetem egyik laborjában készült 310 képpárból keletkezett 3D illetve 2D térkép.

pozíciója. Itt minden mérés esetében a kiindulási és a célpozíció összehasonlítása volt lehetséges. A robot egy teljes kört járt be minden esetben, így az első és utolsó pozíciója és orientációja közel megegyezik. Az első és utolsó pozíció közötti ofszet távolságot az ISVD algoritmussal számoltam ki. Ezért a teljes bejárt területen összegyűlt abszolút transzlációs és rotációs hiba vizsgálható, bár a pontos útvonal nem ismert.

A következő ábrán (5.2. ábra) az egyik ilyen mérési adatsor három, illetve két-dimenziós rekonstrukciója látható. A rekonstrukció a saját alkalmazással készült, majd az alkalmazásban lett megjelenítve. Az adatsoron a egy teljes kört járt be a jármű laboratóriumban. 310 helyzetben készült kép és körülbelül 10 m a bejárt út teljes hossza. Mivel a robot közel vízszintes terepen haladt, így a függőleges irányú halmozott elmozdulás hiba, valamint a roll (ϕ) és a pitch (θ) értéke a vártnak megfelelően lényegesen kisebb, mint a sikban való elmozdulás kétirányú halkmozott hibája, illetve a yaw (ψ) halmozott eltérése. Az összes akkumulálódott abszolút transzlációs és rotációs hiba az 5.3. táblázatban látható.

Készültek további adatsorok az egyetem laborjában, ahol a bejárt út szintén zárt hurok volt, valamint készült egy adatsor, ahol a szenzort kézben tartva 360 fokban körbe



5.3. ábra. Az egyetemen készült két adatsoron futtatott eredmény. Bal oldalt egy a labor háromdimenziós rekonstrukciója, ahol a robot egy zárt hurkot járt be. Jobb oldalon egy irodában készült rekonstrukció, ahol a szenzor kézben tartva, 360 fokban lett körbe forgatva.

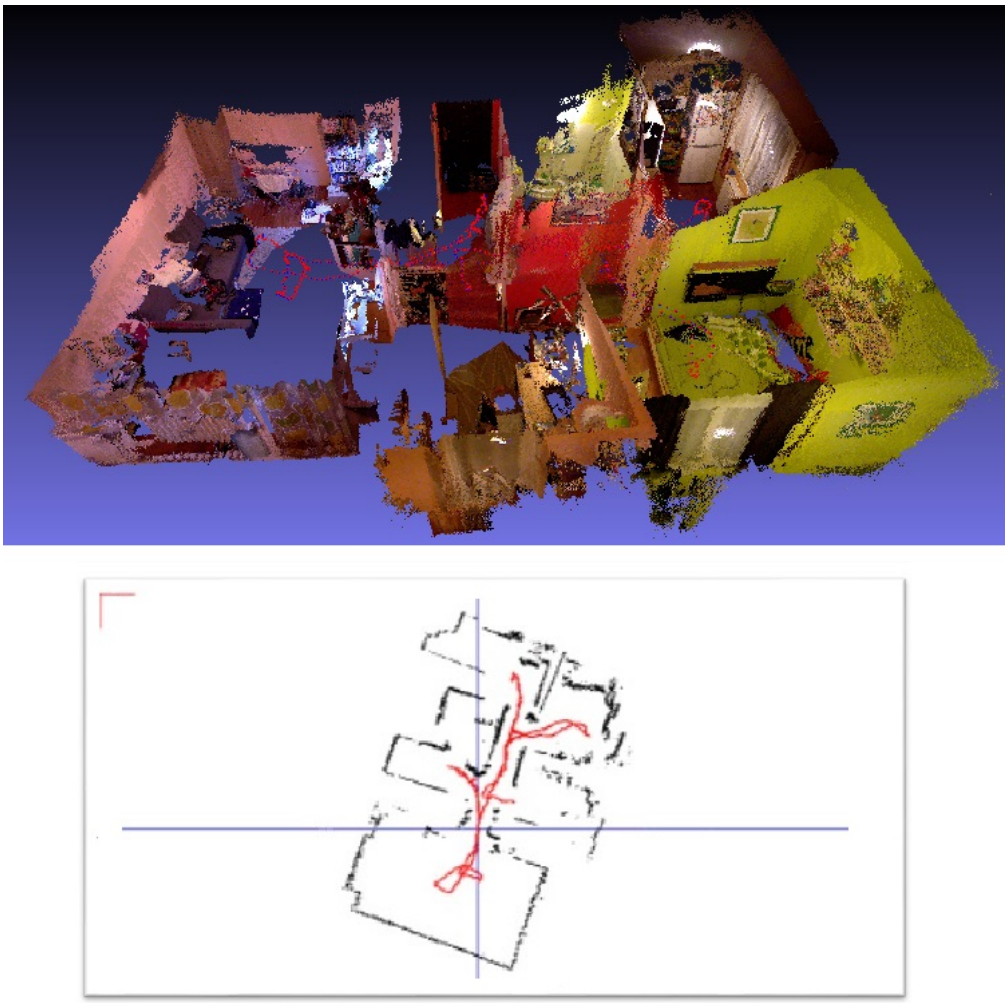
forgatva rögzítettem a környezetet. (5.3. ábra).

Az ismertetett algoritmust további környezetekben is teszteltem. Egy mérés során egy $50m^2$ alapterületű lakás két, illetve háromdimenziós rekonstrukciója valósult meg. Az így elkészült rekonstrukció eredménye a 5.4. ábrán látható. A mérés során a szenzor mozgatása kézben történt, melynek során sorra jártam be az egyes szobákat, helyiségeket, majd az összegyűjtött adatokat felhasználva az alkalmazásom elkészítette a bejárt terület térképét és 3D rekonstrukcióját.

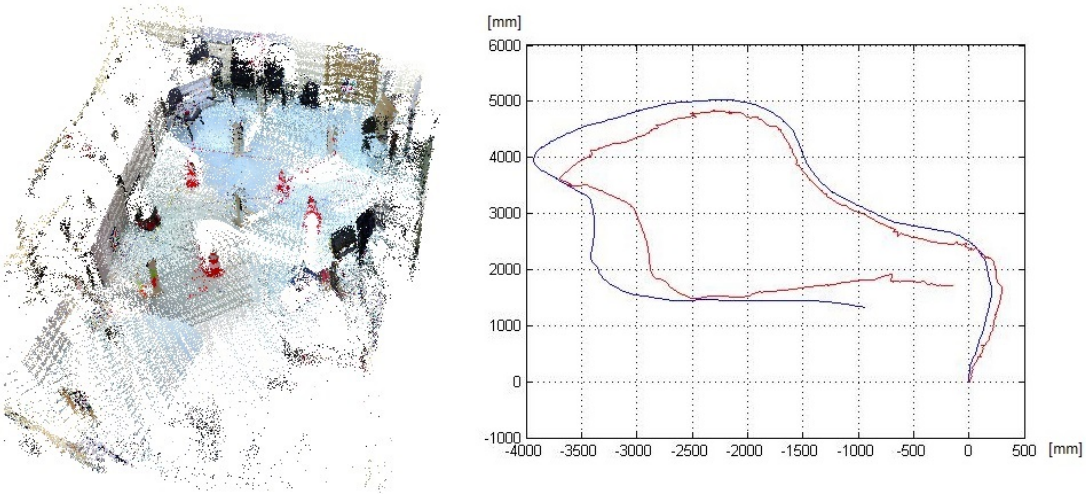
5.6. Poznani egyetem adatsorán történt tesztelés

Ponznani Egyetemen [34] készített adathalmazok esetén rendelkezésre áll minden képpárhoz (Kinect szenzorral készült mélységi és színes kamerakép) a robot valós térbeli pozíciója. Az egyetemen 4 darab mérési sor készült egy WiFiBot-ra szerelt Kinect szenzorral, a jármű pozícióját 5 darab plafonra szerelt kamera rögzítette. Az adathalmaz tartalmazza a Kinect szenzor eredeti színes és mélységi képeit, a kamera paramétereit és a robot mozgását monitorozó mérőrendszer adatait. Az regisztrálás során a képek 100 ms-enként készültek.

A trajectory1 [34] adatsoron különböző beállítások mellett 300 kép illesztése során végeztem összehasonlító méréseket. A tesztelés során a SURF és ORB jellemződetektorok melletti paraméter volt, hogy hány korábbi képhez történjen a jelenlegi mérés illesztése. Az illesztések átlagos eredményét a 5.6. ábra szemlélteti, ahol látható hogy az ORB



5.4. ábra. 2D és 3D rekonstrukció egy $50m^2$-es lakásról. A teljes mérés során 500 képpár készült a Kinect szenzorral.

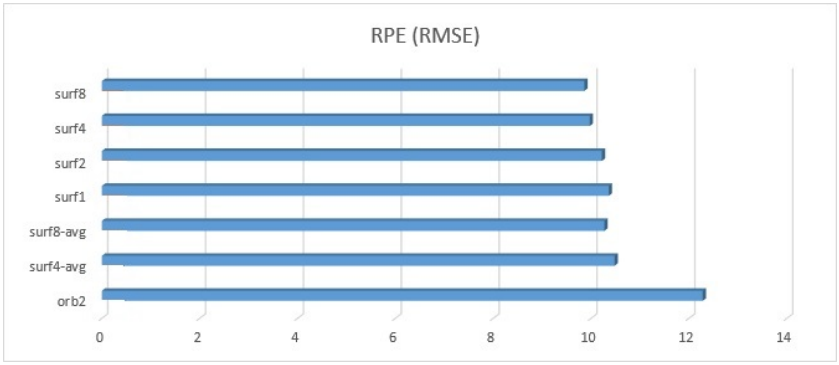


5.5. ábra. Poznani egyetem trajectory1[34] adatsorából készült saját háromdimenziós rekonstrukció az első 700 képből. A jobboldali képen kék színnel a valós elmozdulás, piros színnel a becsült elmozdulás látható. Az algoritmus az első 700 képet dolgozta fel, a SURF jellemződetektor használatával és minden új képet az előző 4 darab képhez illesztve. LC algoritmus és DIF szűrő nem volt használva ebben az esetben.

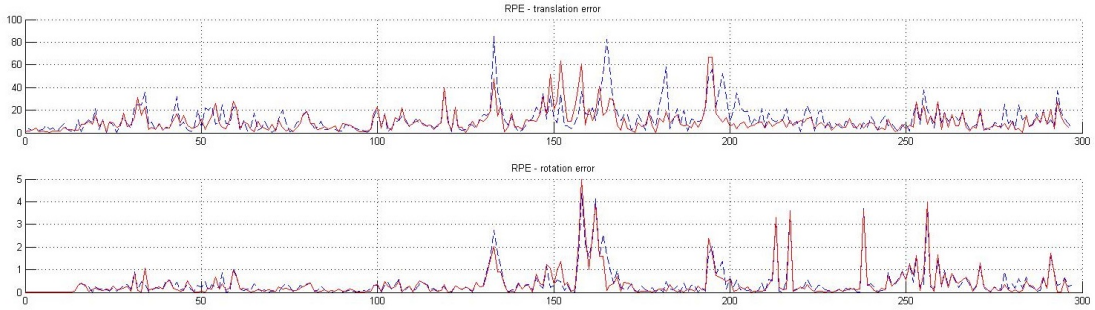
jellemződetektor esetén volt a legnagyobb a hiba, míg a SURF jellemződetektorok esetén kissebb a hiba nagysága. A 5.7. ábra és 5.8. ábra részletesen, képpáronként mutatja az relatív elmozdulási hibákat, ahol látható hogy az ORB jellemződetektor esetén voltak nagyobb kiugrások a grafikonon, míg a SURF jellemződetektorok esetén kissebb vot a hiba kiugrásának nagysága.

Az orb2 adatsor esetén két korábbi méréshez történt illesztés és az ORB jellemződetektort használta az algoritmus. A surf1, surf2 surf4, és surf8 esetében 1, 2, 4 és 8 korábbi méréshez történt illesztés a SURF jellemződetektor felhasználásával, valamint a több méréshez történő illesztés esetén az egyes transzformációk becsült elmozdulás értékei súlyozottan kerültek beszámításra (4.1). A surf4-avg és surf8-avg adatsor esetén a SURF jellemződetektorral 4 és 8 korábbi méréshez történt az illesztés, de az egyes illesztések során sima átlagolással került meghatározásra a transzformáció elmozdulása. Látható, hogy az ORB jellemződetektorral szemben a SURF jellemződetektor jobb eredményt ad. A korábbi mérésekhez történő illesztés számának növelésével tovább növelhető a pontosság. A SURF detektor és legalább 4 és 8 korábbi méréshez történt illesztés esetén a legjobb eredmény akkor született, amikor a transzformációk a 7 algoritmus szerint súlyozott átlaggal (4.1) kerültek beszámításra.

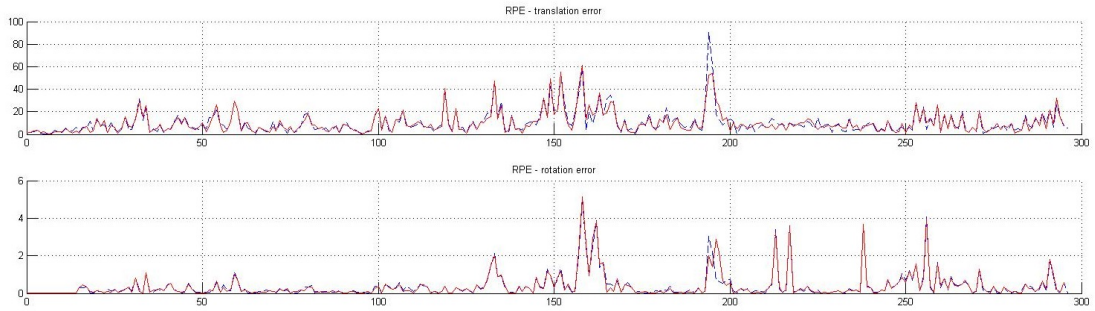
A trajectory1 [34] adatsoron az első 500 kép illesztését végeztem el, akárcsak Kraft-ék



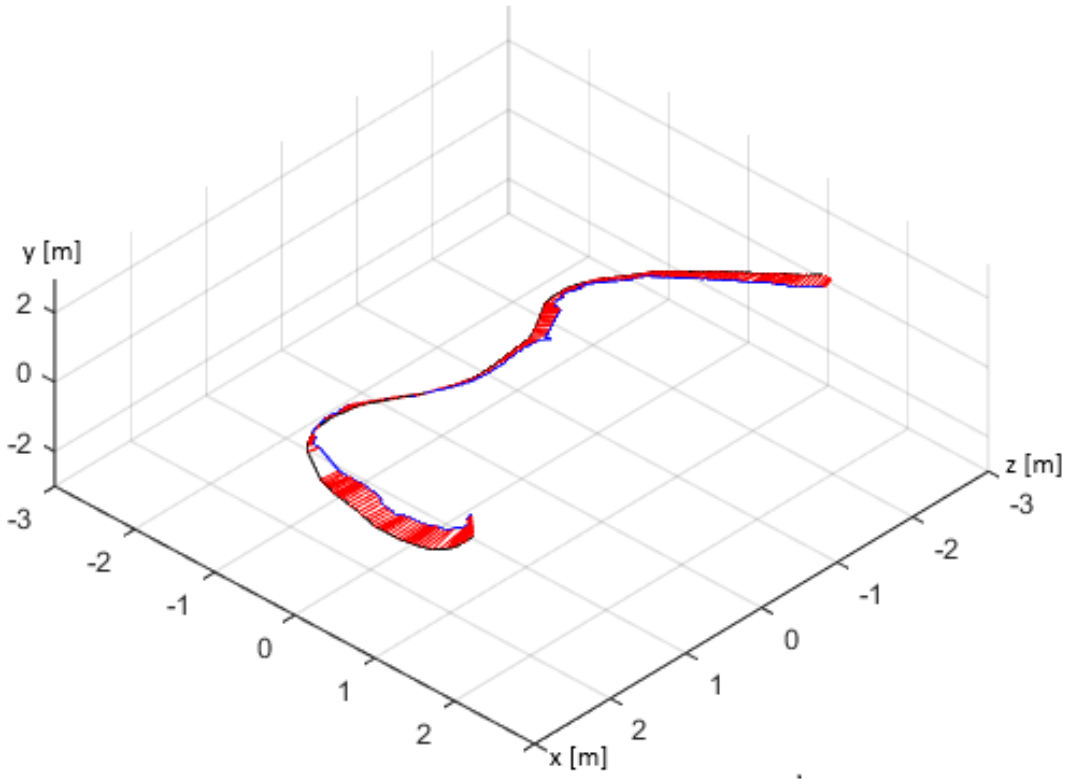
5.6. ábra. A trajectory1[34] adatsoron futtatott illesztések eredményei. Az ábra a különböző jellemződetektorok esetén mutatja az elmozdulások Átlagos Relatív Pozíció Hibáját (RMSE).



5.7. ábra. A trajectory1[34] adatsor esetében a SURF és az ORB jellemződetektorok esetén kapott relatív illesztési hiba. Kék színnel a ORB jellemződetektor, piros színnel az SURF jellemződetektor esetében látható az eredmény. Mindkét esetben 2 korábbi méréshez történt az illesztés.



5.8. ábra. A trajectory1[34] adatsor esetében a SURF jellemződetektorok esetén kapott relatív illesztési hiba. Kék színnel egy korábbi méréshez, piros színnel 4 korábbi méréshez történt az illesztés.



5.9. ábra. A trajectory1 adatsoron végzett mérés eredménye a saját módszerrel, az első 500 kép esetén. Feketével a valós, kékkel a becsült útvonal, pirossal a hiba nagysága. Az eredmények méterben vannak megadva.

csapata egy mérés során [71], [72]. Az eredmény a 5.9. ábrán látható.

5.7. Eredmények értékelése TUM adathalmazokkal és összehasonlítás más rendszerekkel

Ebben a fejezetben az algoritmusom tesztelésének bemutatása következik, ahol a TUM adathalmazait használtam a minőség meghatározására [33], mivel ezekkel széles körben tesztelik a SLAM algoritmusokat és értékelik minőségüket. A Müncheni Műszaki Egyetemen készült adathalmazok több csoportra bonthatók. Készültek adathalmazok, amikor a Kinect szenzort kézben mozgatták, illetve olyanok is, ahol egy robotautóra volt felszerleve. Statikus környeztben, valamint dinamikusan változó feltételek között

is rögzítettek jeleneteket (többen sétáltak a szenzor előtt). Az adatbázisban találhatók textúrában gazdag és szegény képek is. A TUM adathalmazokat sok kutatás használja a saját rendszerének pontosság vizsgálatához, így célszerűen én is ezeket az adathalmazokat választottam az összehasonlíthatóság miatt.

A fejezetben először azt mutatom be, hogy ezeket az adathalmazokat felhasználva a saját módszerem minősége miként fejlődött a korábbi verziókhoz képest (5.7.2. fejezet), majd pedig más szerzők eredményeihez hasonlítom az általam elért pontossági értékeket (5.7.3., 5.7.4. fejezetek). Eredményeim minőségét két csoportban értékelem más hasonló feladatot ellátó rendszerek jellemzőihez viszonyítva. Az első csoportban a statikus környezetben készült adatsorok ATE értékét vetettem össze Guclu et al. csapatának munkájával, az Extended RGBD SLAM rendszerrel [73], Whelan et al. [9], Liu et al. [11], Endres et al.(RGB-D SLAM) [13] és Stückler et al. MRSMap rendszerével [14]. A második csoportban a dinamikus környezetben készült adatsorok ATE értékét táblázatos formában vizsgáltam a Co-fusion [20] és StaticFusion [21] módszerekkel, Hachiuma-ék munkájával [17], Yangdong-ék [18] és Rongsong-ék eredményeivel [19].

A táblázatok több különböző adatsor eredményeit mutatják, amik a TUM/fr1 és fr2 és fr3-as adathalmazok közül lettek kiválasztva. A látható paraméterek ATE RMSE értékek [70], méterben vannak megadva. A kiválasztott adatsorok egy részében található csak zárt hurok. Példaként az 5.10. ábra mutatja be az fr1/desk adatsor 3D rekonstrukcióját a saját módszeremmel.

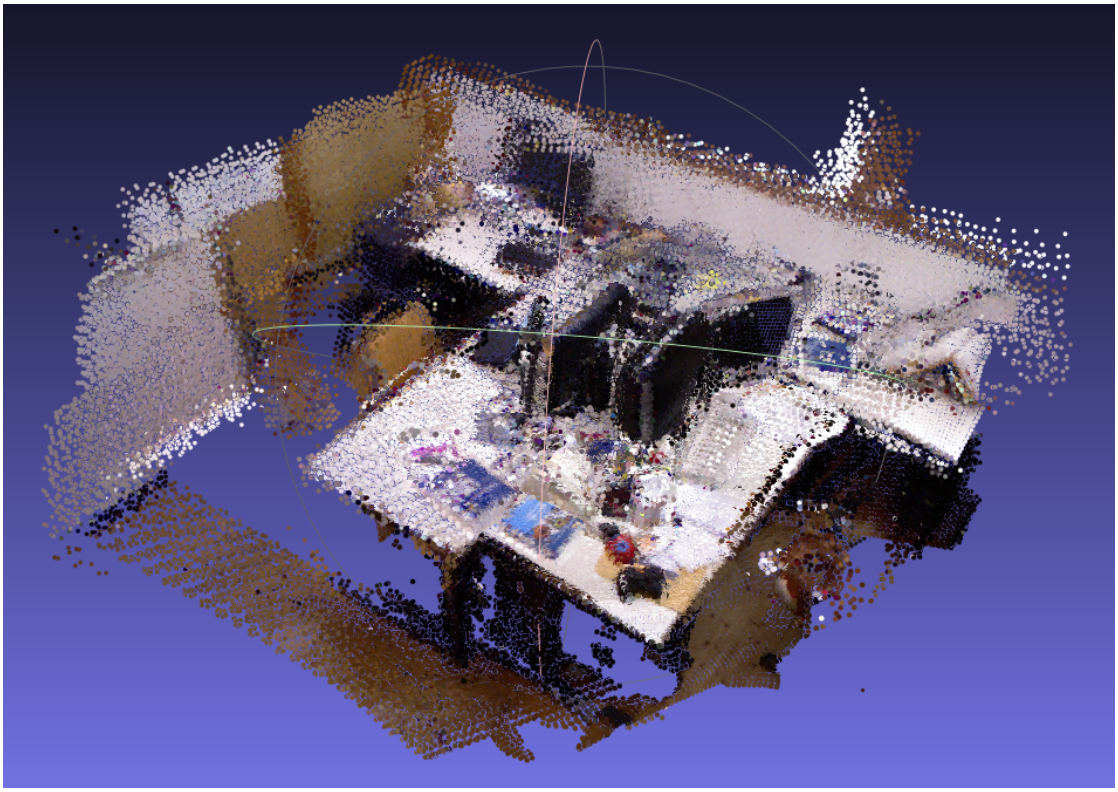
5.7.1. Saját rendszer pontosságának növelése

Az 5.4. táblázatban az algoritmusom jelenlegi illetve egy korábbi változata szerepel. A TUM fr1/desk adathalmazon készült korábbi eredményeim [32], [35] kerültek összehasonlításra a jelenlegi, pontosabb eljárásommal.

Látható hogy a fr1/desk 2018-as változatában [32] még Loop Closure algoritmus nem volt és jóval pontatlanabb a jelenlegi értékeknél, akárcsak a következő eredmény [35], ami már tartalmazta a LC eljárást is. A legpontosabb eredményt a mostani verzió szolgáltatta, ahol a továbbfejlesztett SLAM módszerem (7. ISVD algoritmus) esetén a szenzor adatait feltételes átlagoló szűrővel használom (4. DIF algoritmus) és a zárt hurok (LC) detektálás is folyamatosan működik.

5.7.2. Saját algoritmus pontossága

Az 5.5. táblázatban az ISVD algoritmusom pontosságának vizsgálatához az ATE értékeket vettem alapul, hiszen ezt a mérőszámot használja nagyon sok hasonló kutatás is. Vizsgáltam



5.10. ábra. $fr1/desk$ adatsor 3D rekonstrukciója

5.4. táblázat. A saját algoritmus Absolut Trajectory Error (ATE RMSE) értéke az fr1/desk adathalmazán vizsgálva, bemutatva az eljárásom fejlődését. Minden érték méterben van megadva.

	2018 surf4 [32]	ISVD+LC surf-4 [35]	ISVD+LC surf-4	ISVD+LC+DIF surf-4
fr1/desk	0,0907	0,0628	0,03	0,0298

5.5. táblázat. ISVD algoritmus pontosságának vizsgálata. A táblázatban az ATE értékek méterben vannak megadva. Vizsgáltam az ISVD illesztési algoritmust önmagában és Loop Closure algoritmussal illetve DIF szűrővel kiegészítve is. Az algoritmus futtatásakor az MP paraméter 4 volt minden esetben. A legjobb eredményeket félkövérrel jelöltem a táblázatban.

Adathalmaz	ISVD	ISVD+LC	ISVD+LC+DIF
fr1/rpy	0,0466	**0,026**	0,032
fr1/xyz	0,0232	**0,014**	0,0148
fr1/360	0,1028	**0,0683**	0,0757
fr1/desk	0,0455	0,03	**0,0298**
fr1/room	0,2727	0,17551	**0,1403**
fr1/desk2	0,0847	**0,0534**	0,0535
fr1/plant	0,0517	**0,032**	0,0338
fr2/xyz	0,0776	0,0506	**0,0155**
fr2/desk	0,3431	0,217	**0,1039**
fr2 pioner 360	0,5301	0,4069	**0,3877**
fr2/pioner slam	0,5569	0,4421	**0,3571**
fr2/pioner slam 2	1,2	**1,1754**	1,311
fr2/pioner slam 3	0,471	**0,4558**	0,9211
fr1 teddy	0,1414	0,1432	**0,1383**
fr3 long	0,4502	0,2267	**0,11**
fr2/flowerbouquet	0,1688	0,1682	**0,1202**
fr2/metallic_sphere	0,7879	**0,7785**	0,8273
fr3/walking static	0,0391	0,0268	**0,0252**
fr3 walking xyz	0,2696	0,1184	**0,09**
fr3/walking halfsphere	0,4286	**0,1044**	0,1063
fr3/sitting xyz	0,0716	0,0468	**0,0324**
fr3/sitting halfsphere	0,0716	0,0524	**0,0499**

az ISVD illesztési algoritmusom pontosságát önmagában, Loop Closure algoritmussal, illetve DIF szűrővel kiegészítve is. Az algoritmus futtatásakor az MP paraméter (hány korábbi képhez történik az illesztés) 4 volt minden esetben.

Az eredményekből látszódik, ha csak az ISVD algoritmussal történt a mozgás becslése, akkor kaptam a legrosszabb eredményt a használt adathalmazok esetében. A Loop Closure algoritmussal kiegészítve minden esetben jobb eredményt kaptam, akár nagyobb zárt hurkokat, akár csak kisebb zárt hurkokat tartalmazott az adathalmaz.

A DIF szűrővel (mélységi képen történt szűrés, 4. algoritmus) kiegészítve a LC algoritmust, bizonyos esetekben tovább javult a pontosság. A vizsgált 22 darab adathalmaz esetén a DIF szűrő 13 esetben adott pontosabb eredményt, mint amikor nem használtam a szűrőt. A DIF szűrőt használva az esetek 59%-ában volt jobb eredmény a véletlenszerűen kiválasztott adathalmazok esetében.

A tesztek során használt adathalmazok 2D-s generált térképeit mutatja be az 5.11. ábra, az x és y tengelyre levetítve. Az ábrákon a végső rendszer, vagyis az ISVD+LC+DIF algoritmus eredménye látható, kilenc TUM adatsor esetén. Feketével a valós útvonalat, kékkel a rendszerem által becsült útvonal látható. A becsült és a valós útvonal közti hiba pirossal látható az ábrákon.

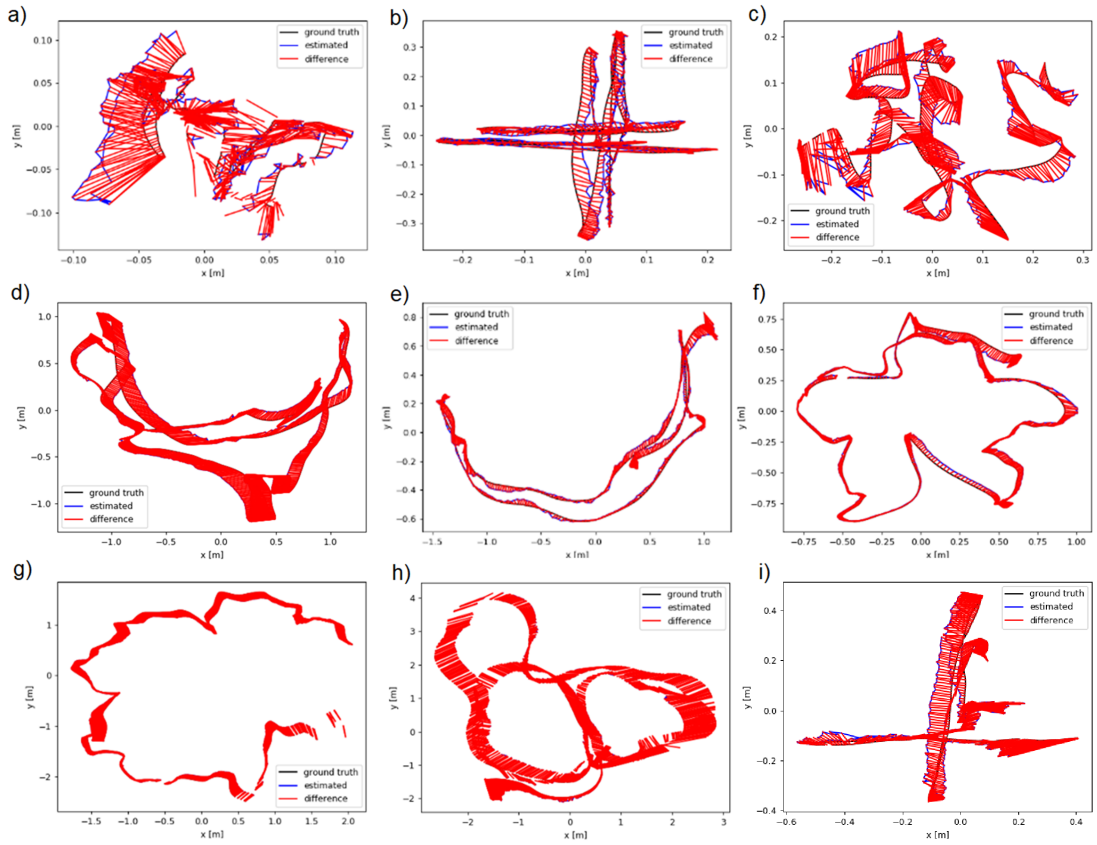
5.7.3. Algoritmus pontosságának összehasonlítása más rendszerekkel, statikus környezetben

Egyik összehasonlításnak olyan adathalmazokat választottam ki a TUM adatbázisból, ahol statikus környezetben készültek a felvételek. Ezekben az adatsorokban nem voltak mozgó tárgyak, személyek a rögzítés idején, csak a szenzort mozgatták a statikus környezetben.

Ezeket az adathalmazokat Guclu et al. csapatának munkájával [73], Whelan et al. [9], Liu et al. [11], Endres et al.(RGB-D SLAM) [13] és Stückler et al. MRSMap rendszerével [14] hasonlítottam össze, aminek az eredményét az 5.6 táblázat összegzi. Az összehasonlításhoz legtöbbször az eredeti cikkekből vettem az eredményeket, ahol nem ott jelöltem melyik másik szerző cikkében volt az adott módszerrel elvégezve a tesztelés.

Az első oszlop az adathalmazok megnevezését tartalmazza, a másodikban a saját legjobb eredményt mutatom be, a további oszlopokban pedig a szerzők eredményei jelennek meg. Stückler at al. MRSMap rendszerének [14] futási eredményeit [9] publikációban szereplő értékek, míg az (RGB-D SLAM) [13] eredményeit a szerzők futtatásai, valamint [73] és [11] alapján mutatom be a táblázatban. Félkövérrel kiemeltem, hogy az adott adathalmazon melyik megközelítés szolgáltatta a legjobb eredményt.

Öt esetben, az fr1/rpy, fr1/xyz, fr1/360, fr2/flowerbouquet és az fr2/metallic sphere



5.11. ábra. Végső rendszerem, azaz az ISVD+LC+DIF algoritmusok futási eredményei az a) fr1/rpy, b) fr1/xyz, c) fr1/360, d) fr1/room, e) fr1/desk2, f) fr1/plant, g) fr2/desk, h) fr2/pioner slam 2 és i) fr3/walking xyz adathalmazokon. Feketével a valós útvonal, kékkel a becsült útvonal és pirossal a hiba látható, minden tengelyen méterben vannak az értékek megadva.

esetén kaptam jobb ATE értéket a vizsgált hasonló munkáknál. Ezekben az esetekben kis mértékben jobb eredményt kaptam és a többi tesztesetben is nagyságrendileg megegyeztek az általam kapott ATE értékek a vizsgáltakkal.

5.7.4. Algoritmus pontosságának összehasonlítása más rendszerekkel, dinamikus környezetben

Második összehasonlításnak olyan adathalmazokat választottam ki a TUM adatbázisból, ahol dinamikus környezetben készültek a felvételek. Ezekben az adatsorokban voltak mozgó tárgyak, személyek a rögzítés idején, így nem csak a szenzort mozgatták adott környezetben, hanem a környezet egy része is változott.

Ezeket az adathalmazokat felhasználva munkámat a Co-fusion [20], a StaticFusion [21] módszerekkel, Hachiuma-ék eredményeivel [17], Yangdong-ék [18] és Rongsong-ék munkájával [19] hasonlítottam össze, aminek az eredményét az 5.7 táblázat összegzi. Az összehasonlításhoz legtöbbször az eredeti cikkekből vettem az eredményeket, ahol nem ott jelöltem melyik másik szerző cikkében volt az adott módszerrel elvégezve a tesztelés.

Az eredményekből látható, hogy az eljárásom ezt a környezetet is kezelni tudja. A kiválasztott adatsorok esetében, az összehasonlításra használt munkák mellett jobb eredményt ebben az esetben nem adott a rendszerem, azonban nagyságrendileg az eredményeim a vizsgált rendszerekkel összevethető szintűek.

5.6. táblázat. A saját algoritmus és a hasonló rendszerek Absolut Trajectory Error (ATE RMSE) értékei, fr1, fr2 és fr3 adathalmazokon vizsgálva, statikus környezetben. Minden érték méterben van megadva.

Adathalmaz	Saját jobb	Ext. RGBD SLAM [73]	Whelan et al.[9]	Liu et al. [11]	RGBD SLAM [13]	MRSMap [14]
fr1/rpy	**0,025**		0,028			0,027* [9]
fr1/xyz	**0,013**	0,014	0,017	0,013	0,014* [11]	0,013* [9]
fr1/360	**0,056**	0,075				
fr1/desk	0,026	**0,022**	0,037	0,064	0,026	0,043* [9]
fr1/room	0,14		0,075		0,087	**0,069*** [9]
fr1/desk2	0,049	**0,034**	0,071			0,049* [9]
fr1/plant	0,034	0,068	0,047			**0,026*** [9]
fr2/xyz	0,0155			0,015	**0,008*** [11]	
fr2/desk	0,12	0,090	**0,034**		0,057	0,052* [9]
fr2/ flowerbouquet	**0,1202**	0,137			0,131* [73]	
fr2/metallic sphere	**0,7785**	0,914			1,099* [73]	
fr2/pioner slam	0,36	**0,349**			0,367* [73]	
fr2/pioner slam 2	1,1754	0,4			**0,381*** [73]	
fr2/pioner slam 3	0,4558	**0,341**			0,511* [73]	
fr3/long	0,11			**0,028**	0,032* [11]	

5.7. táblázat. A saját algoritmus és hasonló rendszerek Absolut Trajectory Error (ATE RMSE) értékei, fr3 adathalmazokon vizsgálva, dinamikus környezetben. Minden érték méterben van megadva.

Adat halmaz	Saját jobb	Co-fusion [20]	StaticFusion [21]	Hachiuma DF [17]	Yangdong BS+DR [18]	Rongsong DUCK[19]
fr3/walking static	0,0252	0,551 *[17]	**0,014** *[17]	0,036	0,029	0,0235
fr3/walking xyz	0,09	0,696 *[17]	0,127 *[17]	0,085	0,126	**0,0426**
fr3/walking halfsphere	0,1044	0,803 *[17]	0,391 *[17]	**0,072**	0,336	
fr3/sitting xyz	0,0324	0,027 *[17]	0,04 *[17]	0,052	0,045	**0,0203**
fr3/sitting halfsphere	0,0499	**0,036** *[17]	0,04 *[17]	0,041	0,037	

6 Összefoglalás

6.1. Összegzés és tézisek

Másfél évtizede foglalkozom mobil robotok fejlesztésével. Tevékenységem mindig kétirányú volt, megterveztem és elkészítettem a rendszerekhez szükséges elektronikai terveket, áramköröket, illetve kifejlesztettem nemcsak az ezek használatához szükséges programokat, hanem magas szintű navigáló alkalmazásokat is terveztem és megvalósítottam. Az első ilyen mobilrobot 2008-ban a Magyarok a Marson című versenyen II. helyezést ért el. Továbbfejlesztett változata a 2009-es Országos Tudományos Diákköri Konferencia, Műszaki tudományok szekciójában első helyezésben részesült ("Mavridisz Vaszilisz, Somlyai László, Gál Béla: Lézerszkennerrel támogatott körbelátórendszer önjáró roboton"). Fő szerepem a hardvert működtető elektronika megtervezésében és az azt vezérlő firmware elkészítése volt.

A robotos rendszerünket lényegesen továbbfejlesztve mutattuk be a 2011-es OTDK Informatkatudományi szekcióban, ahol dolgozatunkat (Csaba György, Somlyai László: Gépi látáson alapuló akadályelkerülés) II. helyezéssel értékelték.

A rendszerünket egy távirányítós autóra építettük fel. Az autó eredeti vezérlő rendszere helyett két elektronikai kártyát terveztem, aminek a segítségével az autó számítógépről irányítható volt. Ezek a CAN bridge nevű kártya (6.1.3. melléklet) és a motormeghajtó modul (6.1.3. melléklet).

Az OTDK-n elért eredmény mellett a rendszerünket két magyar nyelvű előadáson [22], [23] és egy nemzetközi konferencián [24] is bemutattuk.

A kisméretű robotok navigációja és vezérlése mellett dolgoztam számos másik projektben, így például egy kisméretű GPS által vezérelt robotrepülő kommunikációs protokolljának kifejlesztésében [25]. Terveztem merevszárnyú robotrepülőt vezérlő elektronikákat és készítettem egy antennaforgató rendszert robotrepülőkhez, ahol a GPS rendszer pontosságával megismerkedtem. Majd készítettem egy légköri paramétereket mérő adatgyűjtő rendszert [26], amit kis méretű robotrepülőn, vagy akár robotautón elhelyezhető.

Később a kutatási tevékenységem fokozatosan a navigáció terüetére és a navigációhoz

szükséges adatok gyűjtésére irányult. Így összehasonlítottuk a saját strukturált megvilágítást használó érzékelőnket, amit korábban fejlesztettünk ki, a Kinect szenzorral. Itt vizsgáltam a Kinect szenzor előnyeit és pontosságát [27].

A robotautó mozgásbecsléséhez később áttértem a Kinect szenzor használatára, hiszen ezzel nagyobb pontosság és részletgazdagabb térkép építhető fel. Ezért a korábban kifejlesztett lézerszkennert felváltottam egy RGB-D szenzorra. A szenzort kalibráltam, majd elhelyeztem a saját robotautómra [28]. Az összeépített rendszer segítségével az Egyetem laborjaiban készítettem számos off-line adathalmazt a Kinect szenzorral, amit később az algoritmusaim teszteléséhez tudtam felhasználni.

Kezdeti próbálkozásként a szenzor mozgásának becslésére egy SVD felbontáson alapuló iterativ algoritmussal kísérleteztem. Néhány kiválasztott jellemző pont pár segítségével az SVD felbontást használva határoztam meg két egymás utáni pozició közötti relatív elmozdulást [29]. Ennél az algoritmusnál a megvalósított rendszer még nagy futási idővel, és viszonylag nagy pontatlansággal is rendelkezett.

Az eredeti SVD felbontást használó algoritmusnak a továbbfejlesztésével jelentős javulást értem el a futási idő és pontosság tekintetében is. Ezt a munkámat ISVD algoritmusnak neveztem el és egy konferencián [30] és egy folyóiratban [31] is publikáltam.

A kezdeti keretrendszemet továbbfejlesztve elkészítettem egy alkalmazást, az ISVD algoritmusom teszteléséhez [32]. Ebben a keretrendszerben a saját laboratóriumi adathalmazokon kívül, két külső adathalmazt is vizsgáltam (TUM [33], POZNAN [34]). A keretrendszer segítségével összehasonlítottam a saját munkám pontosságát más kutatások pontosságával.

A rendszer egy későbbi változata pontosabb becslést ad az elmozdulásra és a működése is robusztusabb a korábbi rendszernél, a továbbfejlesztett illesztési eljárásnak és a valós idejű Loop-closure algoritmusnak köszönhetően [35]. Ennél a rendszernél nem egy korábbi képhez történt az illesztés, hanem állítható számú korábbi képhez illesztettem az utolsó mérést.

Végül a jelenlegi rendszerem pontosságát a több korábbi képnek már nem csak átlagolt, hanem súlyozott illesztésével értem el. Ezt az algoritmust ISLAM-nak neveztem el [36]. Itt vizsgáltam a saját rendszerem és számos más rendszer közötti pontosságot 20 online elérhető adathalmazt felhasználva. Vizsgáltam a rendszerem működőképességét statikus és dinamukus környezetben is, illetve a korábban kifejlesztett DIF szűrömet (4) is teszteltem. A vizsgált adathalmazok esetében a rendszer átlagosan 10 képet tudott feldolgozni másodpercenként.

6.1.1. 1. tézis: SVD alapú iterativ SLAM eljárást fejlesztettem RGB-D kamerák által szolgáltatott pontfelhők illesztésére.

Kifejlesztettem egy SVD-n alapuló iteratív illesztési eljárást (ISVD), amely az illesztésre szánt pontfelhők jellemzőpontjaiból több lépésben távolítja el a kiugró értékeket. A kidolgozott módszer minden iterációban megvizsgálja a jellemző pontpárok relatív hibáját, a nagy hibával jellemezhető pontpárokat eltávolítja az illesztendők halmazából, majd fokozatosan csökkenti a megengedett hibaértéket a következő párosítási iterációhoz. Az iterációk végén a legjobb illesztést biztosító pontpárokból kerül meghatározásra az illesztést biztosító transzformáció (becsült elfordulás és elmozdulás).

A tézist alátámasztó saját publikációk: [30], [31], [32].

Magyarázat

A kifejlesztett algoritmus (ISVD) egy RGB-D kamera használatán alapú háromdimenziós rekonstrukciós és térképépítési eljárás. A rendszer valós időben képes megbecsülni a jármű (mobil robot) mozgását, valamint folyamatosan frissíti a navigáció során a detektált környezet térképét. A működés során csak a szenzor színes kamera képe és pixelszintű mélységi információja kerül felhasználásra.

Az algoritmus olyan háromdimenziós pontfelhőkre támaszkodik, amik között átfedés található, valamint a pontfelhőkhöz kamerakép rendelhető. Az ilyen pontfelhők egymáshoz illeszthetőek, meghatározható ez alapján a szenzor relatív elmozdulása és orientációja. Az elmozdulás becsléséhez az algoritmus a környezet háromdimenziós pontfelhőiben, az egymás után vett képeken az összetartozó jellemzőpontok párjait keresi meg. A különböző időpontokban készült háromdimenziós jellemzőpontok közti transzformáció meghatározása egy SVD algoritmuson alapuló módszerrel történik. Az illesztés többlépéses iteráció során minimalizálja az illesztési hibát és törli a hibásan detektált pontpárokat (6. algoritmus).

Gyakorlati alkalmazás

Az algoritmusomat az Egyetemen készített saját adatsorokon teszteltem. A kapott eredményekben mind a becsült átlagos elfordulás, mind a becsült átlagos elmozdulás érték kisebbnek bizonyultak, mint ha a szakirodalomban hagyományos ICP módszert alkalmaznám (4.1. táblázat).

6.1.2. 2. tézis: Több egymás után rögzített RGB-D pontfelhő illesztésének minőségét figyelembe vevő eljárást fejlesztettem, amely a becsült elmozdulások pontosságát javítja.

Az SVD-n alapuló iteratív illesztési eljárást (ISVD) úgy fejlesztettem tovább, hogy a módszer több legutóbbi illesztésre szánt pontfelhő jellemzőpontjaiból határozza meg iteratívan az illesztést biztosító transzlációt és orientációt. Az egyes illesztések pontosságára, minőségére vonatkozóan paramétert határoztam meg, és a végső elmozdulás becslésébe ezen paraméterek súlyozásával épül be a részeredmény.

A tézist alátámasztó saját publikáció: [36].

Magyarázat

A kifejlesztett algoritmus (ISLAM) egymás után készített háromdimenziós pontfelhőkre épül, és feltételezi, hogy közöttük átfedés van. A pontfelhők között illesztést biztosító transzformációt az ISVD algoritmus segítségével határozza meg a módszer több szálon párhuzamosan futtatva az egyes illesztéseket. Az illesztés két lépcsőben valósul meg. Először a két színes képen határoz meg jellemző pontokat, majd pedig ezeket illeszti úgy, hogy a kapott homográfia transzformáció a megvizsgált szituációk közül a legtöbb illeszkedő pontpárt biztosítsa (8. algoritmus). A második részben az egyes illesztések hibájából az illesztés minőségére vonatkozóan jellemzőt határoz meg módszerem, majd pedig ezekkel a súlyokkal javítja az elmozdulás becslését (7. algoritmus).

Az algoritmus jelentősen javuló pontossági értékeket szolgáltat, ami minden esetben hasonló nagyságú, mint a szakirodalmi módszerek, bizonyos teszthalmazoknál pedig meg is haladta azok jellemzőit (5.6. táblázat).

Gyakorlati alkalmazás

Az algoritmusomat a TUM adathalmazon teszteltem. A kapott eredményeket összehasonlítottam 20 adathalmazon, ahol voltak statikus és dinamikusan változó környezetek is. A becsült átlagos elmozdulás érték kisebbnek bizonyultak a következő benchmark adathalmazokon: $fr1/rpy$, $fr1/xyz$, $fr1/360$, $fr2/flowerbouquet$, $fr2/metallicsphere$.

6.1.3. 3. tézis: RGB-D pontfelhők szűrésével és zárt hurok (LC) detektálására kifejlesztett módszerrel teljes rendszert fejlesztettem guruló robotok SLAM problémájának megoldására.

Rendszerem Front-end modulja saját zárthurok detektáló (LC, vagy Loop Closure) algoritmust futtat, ami folyamatosan figyeli az aktuális kép és a kulcsképkockák közti egyezéseket. Ha egyezést talál a rendszer egy korábbi kulcsképpel, akkor az aktuális pozíció pontossága tovább javítható a korábbi kulcskép pozícióját felhaszálva (4.6. ábra). Amennyiben az RGBD szenzor méréseit feltételes átlagoló szűrővel javítjuk (DIF szűrő, 4. algoritmus), akkor a SLAM rendszerem minősége összevethető eredményt szolgáltat több hasonló rendszer jellemzőivel.

A tézist alátámasztó saját publikációk: [35], [36].

Magyarázat

A LC eljárás az aktuális kép jellemző leíróit a korábbi kulcsképek jellemző leíróival összehasonlítva határozza meg az egyezést. Ha egyezést talál egy korábbi képpel, akkor a többlépéses illesztéssel (7. algoritmus) megpróbál relatív elmozdulást meghatározni a kulcskép és az aktuális kép között. A rendszer nem vizsgál meg minden aktuális képet, csak akkor fut le a Loop-closure algoritmus, ha az adott kép megfelelő élességgel rendelkezik. Az élesség megállapítása a korábbi 20 kép jellemző leíróinak alapján történik. Ha a legutóbbi képeken talált jellemző leíró pontok számának a mediánértékénél 1,2-szer több az aktuális kép jellemző leírók száma, akkor ez a kép kevésbé elmosódott és részletgazdagabb, mint a többi (4.7. ábra). Az LC algoritmus futási ideje jelentősen csökken, mert kis számú kulcsképet kell megvizsgálni a többlépéses illesztéssel, aminek nem elhanyagolható a futási ideje a jellemző párok keresése, illetve a többlépéses illesztés miatt.

Ha a rendszert az RGBD szenzor mérészajának csökkentésére kifejlesztett feltételes átlagoló szűrőt alkalmazza (4. algoritmus), akkor guruló robotok navigácijára alkalmas SLAM rendszerem pontossága tovább javul.

Gyakorlati alkalmazás

A LC algoritmus egyetlen kivétellel tovább javította a SLAM megoldáom pontosságát. A vizsgált 22 darab adathalmazból 13 esetben pontosabb lett az eredmény, amikor a DIF szűrővel és a zárt hurok detektáló eljárással együtt teszteltem SLAM rendszerem (5.5. táblázat).

Ábrák jegyzéke

Táblázatok jegyzéke

Algoritmusok jegyzéke

Hivatkozások

[1] Kazutoshi Noda és Hidenobu Aizawa. „Indoor environmental monitoring system using a robot vacuum cleaner”. *Sensors and Materials* 32.3 (2020), 1133–1140. old.

[2] C.S. Manikandababu, R Vijayakrishna, R Venugopal és Y Vishnuprasad. „Development of an Autonomous Solar Grass Cutting Robot with a Path Memorizing Mechanism”. *2024 International Conference on Advances in Computing, Communication and Applied Informatics (ACCAI)*. 2024, 1–7. old. DOI: 10.1109/ACCAI61061.2024.10602185.

[3] Davide Scaramuzza és Friedrich Fraundorfer. „Visual odometry [tutorial]”. *IEEE robotics & automation magazine* 18.4 (2011), 80–92. old.

[4] György Csaba, László Somlyai és Zoltán Imre Vámossy. „Mobil Robot Navigation Using 2D LIDAR”. *2018 IEEE 16th World Symposium on Applied Machine Intelligence and Informatics (SAMI) : Dedicated to the Memory of Pioneer of Robotics Antal (Tony) K. Bejczy : proceedings*. 2018, 143–148. old. DOI: 10.1109/SAMI.2018.8324002.

[5] Asus. *Asus Xiton*. URL: http://xtionprolive.com/asus-3d-depth-camera/asus-xtion2 (elérés dátuma 2024. 08. 01.).

[6] Intel. *RealSense*. URL: https://www.intelrealsense.com/ (elérés dátuma 2024. 08. 01.).

[7] Microsoft. „Microsoft Kinect SDK”. *http://www.microsoft.com/en-us/kinectforwindows/ (Visited: 2012. jan. 26.)* (2012).

[8] F. Fraundorfer D. Scaramuzza. „Part I the first 30 years and fundamentals”. *IEEE Robotics & Automation Magazine* 18.4 (2011), 80–92. old.

[9] Thomas Whelan, Michael Kaess, Hordur Johannsson, Maurice Fallon, John Leonard és John McDonald. „Real-time large-scale dense RGB-D SLAM with volumetric fusion”. *The International Journal of Robotics Research* 34 (2015), 598–626. old. DOI: 10.1177/0278364914551008.

[10] Thomas Whelan, Renato Salas-Moreno, Ben Glocker, Andrew Davison és Stefan Leutenegger. „ElasticFusion: Real-time dense SLAM and light source estimation”. *The International Journal of Robotics Research* 35 (2016. szept.), 1697–1716. old. DOI: 10.1177/0278364916669237.

[11] Qiang Liu, Ruihao Li, Huosheng Hu és Dongbing Gu. „Building semantic maps for blind people to navigate at home”. *2016 8th Computer Science and Electronic Engineering (CEEC)* (2016. szept.), 12–17. old. DOI: 10.1109/CEEC.2016.7835881.

[12] Felix Endres, Jurgen Hess, Nikolas Engelhard, Jurgen Sturm, Daniel Cremers és Wolfram Burgard. „An evaluation of the RGB-D SLAM system”. *Proceedings - IEEE International Conference on Robotics and Automation* (2012. máj.), 1691–1696. old. DOI: 10.1109/ICRA.2012.6225199.

[13] Felix Endres, Jurgen Hess, Jurgen Sturm, Daniel Cremers és Wolfram Burgard. „3-D mapping with an RGB-D camera”. *Robotics, IEEE Transactions on* 30 (2014. febr.), 177–187. old. DOI: 10.1109/TRO.2013.2279412.

[14] J. Stuckler és S. Behnke. „Multi-resolution surfel maps ¨ for efficient dense 3d modeling and tracking”. *Journal of Visual Communication and Image Representation* (2014), 137–147. old.

[15] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren és Dieter Fox. „RGB-D mapping: Using depth cameras for dense 3D modeling of indoor environments”. *In Proc. of the Intl. Symp. on Experimental Robotics (ISER)*. 2010.

[16] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren és Dieter Fox. „RGB-D Mapping: Using Kinect-Style Depth Cameras for Dense 3D Modeling of Indoor Environments”. *International Journal of Robotic Research - IJRR* 31 (2012. ápr.), 647–663. old. DOI: 10.1177/0278364911434148.

[17] R Hachiuma, C Pirchheim és D Schmalstieg. „DetectFusion: Detecting and segmenting both known and unknown dynamic objects in real-time SLAM”. *arXiv preprint arXiv:1907.09127* (2019).

[18] Yangdong Liu, Wei Gao és Zhanyi hu. „3D Scanning of High Dynamic Scenes Using an RGB-D Sensor and an IMU on a Mobile Device”. *IEEE Access* PP (2019. febr.). DOI: 10.1109/ACCESS.2019.2900740.

[19] Rongsong Gou, Guangzhu Chen, Chengliang Yan, Xin Pu, Yuanyuan Wu és Yuan Tang. „Three-dimensional dynamic uncertainty semantic SLAM method for a production workshop”. *Engineering Applications of Artificial Intelligence* 116 (2022. nov.). DOI: 10.1016/j.engappai.2022.105325.

[20] Martin Rünz és Lourdes Agapito. „Co-Fusion: Real-time Segmentation, Tracking and Fusion of Multiple Objects”. *In IEEE International Conference on Robotics and Automation* (2017), 4471–4478. old.

[21] R. Scona, M. Jaimez, Y. R. Petillot, M. Fallon és D. Cremers. „StaticFusion: Background Reconstruction for Dense RGB-D SLAM in Dynamic Environments”. *In IEEE International Conference on Robotics and Automation* (2018).

[22] György Csaba, Laszlo Somlyai és Zoltan Vamossy. „Mobil robot fejlesztése strukturált megvilágításon alapuló szenzorok felhasználásával”. *RobotNap 2011 – Robotika Szakosztály Évkönyve, CD ISSN: 2060-5943, Magyarország* (2011).

[23] György Csaba, Laszlo Somlyai és Zoltan Vamossy. „Mobil robot navigációja strukturális megvilágítás valamint Kinect szenzor használatával”. *Informatika a felsőoktatásban 2011 konferencia Debrecen, Magyarország* (2011), 835–842. old.

[24] György Csaba, Laszlo Somlyai és Zoltan Vamossy. „Mobile robot navigation in unknown environment using structured light”. *LINDI 2011 - 3rd IEEE International Symposium on Logistics and Industrial Informatics, Proceedings* (2011), 249–254. old. DOI: 10.1109/LINDI.2011.6031158.

[25] D. Stojcsics és L. Somlyai. „Improvement methods of short range and low bandwidth communication for small range UAVs”. *IEEE 8th International Symposium on Intelligent Systems and Informatics*. 2010, 93–97. old. DOI: 10.1109/SISY.2010.5647224.

[26] L. Somlyai, A. Turóczi és A. Molnar. „Atmospheric Analyser for Mobile Robots”. *13th IEEE International Symposium On Computational Intelligence And Informatics (CINTI 2012)* (2012), 181–185. old.

[27] G. Csaba, L. Somlyai és Z. Vámossy. „Differences between Kinect and structured lighting sensor in robot navigation”. *2012 IEEE 10th International Symposium on Applied Machine Intelligence and Informatics (SAMI)* (2012), 85–90. old.

[28] L. Somlyai és Z. Vámossy. „Map Building with RGB-D Camera for Mobil Robot”. *IEEE 16th International Conference on Intelligent Engineering Systems 2012 (INES 2012)* (2012), 489–493. old.

[29] László Somlyai. „Mobil robot localization using RGB-D camera”. *2013 IEEE 9th International Conference on Computational Cybernetics (ICCC)*. 2013, 131–136. old. DOI: 10.1109/ICCCyb.2013.6617575.

[30] L. Somlyai és Z. Vámossy. „SLAM algorithm for mobile robot localization with RGB-D camera”. *Fluids, Heat and Mass Transfer, Mechanical and Civil Engineering: Proceedings of the 11th International Conference on Fluid Mechanics (FLUIDS '15)* (2015), 89–94. old.

[31] L. Somlyai és Z. Vámossy. „Vision-based mobile robot localisation and mapping system with Kinect”. *International Journal of Systems Applications - Engineering and Development* 10 (2016), 241–246. old. ISSN: 2074-1308.

[32] L. Somlyai, Gy. Csaba és Z. Vámossy. „Benchmark system for novel 3D SLAM algorithms”. *2018 IEEE 16th World Symposium on Applied Machine Intelligence and Informatics (SAMI)* (2018), 131–136. old.

[33] J. Sturm, N. Engelhard, F. Endres, W. Burgard és D. Cremers. „A benchmark for the evaluation of RGB-D SLAM systems”. *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems* (2012), 573–580. old.

[34] Adam Schmidt, Michał Fularz, Marek Kraft, Andrzej Kasinski és Michał Nowicki. „An Indoor RGB-D Dataset for the Evaluation of Robot Navigation Algorithms”. *Prof. of Int. Conf. on Advances Concepts for Intelligent Vision Systems, Lecture Notes in Computer Science* (2013), 321–329. old.

[35] László Somlyai és Zoltan Vamossy. „ISVD-Based Advanced Simultaneous Localization and Mapping (SLAM) Algorithm for Mobile Robots”. *Machines, IF: 2.6 Q2* 10 (2022. jún.), 519. old. DOI: 10.3390/machines10070519.

[36] László Somlyai és Zoltan Vamossy. „Improved RGB D Camera-based SLAM System for Mobil Robots”. *Acta Polytechnica Hungarica, IF: Q2* 21.8 (2024), 107–124. old. DOI: 10.12700/APH.21.8.2024.8.6.

[37] P.J. Besl és Neil D. McKay. „A method for registration of 3-D shapes”. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14.2 (1992), 239–256. old. DOI: 10.1109/34.121791.

[38] François Pomerleau, Francis Colas és Roland Siegwart. „A Review of Point Cloud Registration Algorithms for Mobile Robotics”. *Foundations and Trends® in Robotics* 4.1 (2015), 1–104. old. ISSN: 1935-8253. DOI: 10.1561/2300000035.

[39] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren és Dieter Fox. „RGB-D mapping: Using depth cameras for dense 3D modeling of indoor environments”. *Experimental robotics*. Springer. 2014, 477–491. old.

[40] S Seeger, X Laboureux és G Häusler. „An accelerated ICP-algorithm”. *Lehrstuhl für Optik, Annual Report* (2001), 32. old.

[41] Szymon Rusinkiewicz és Marc Levoy. „Efficient variants of the ICP algorithm”. *Proceedings third international conference on 3-D digital imaging and modeling*. IEEE. 2001, 145–152. old.

[42] Edward Rosten és Tom Drummond. „Machine learning for high-speed corner detection”. *European conference on computer vision*. Springer. 2006, 430–443. old.

[43] David G Lowe. „Distinctive image features from scale-invariant keypoints”. *International journal of computer vision* 60.2 (2004), 91–110. old.

[44] Herbert Bay, Andreas Ess, Tinne Tuytelaars és Luc Van Gool. „Speeded-up robust features (SURF)”. *Computer vision and image understanding* 110.3 (2008), 346–359. old.

[45] Albert S Huang, Abraham Bachrach, Peter Henry, Michael Krainin, Daniel Maturana, Dieter Fox és Nicholas Roy. „Visual odometry and mapping for autonomous flight using an RGB-D camera”. *Robotics Research*. Springer, 2017, 235–252. old.

[46] Luo Juan és Oubong Gwun. „A comparison of SIFT, PCA-SIFT and SURF”. 3. köt. 4. 2009, 143–152. old.

[47] Ethan Rublee, Vincent Rabaud, Kurt Konolige és Gary Bradski. „ORB: An efficient alternative to SIFT or SURF”. *2011 International conference on computer vision*. Ieee. 2011, 2564–2571. old.

[48] K Somani Arun, Thomas S Huang és Steven D Blostein. „Least-squares fitting of two 3-D point sets”. *IEEE Transactions on pattern analysis and machine intelligence* 5 (1987), 698–700. old.

[49] SD Blostein és TS Huang. „Estimating 3-D Motion From Range Data”. *Unknown Host Publication Title*. IEEE, 1984, 246–250. old.

[50] David Cyganski és John A Orr. „Applications of tensor theory to object recognition and orientation determination”. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 6 (1985), 662–673. old.

[51] TS Huang, SD Blostein és EA Margerum. „Least-squares estimation of motion parameters from 3-D point correspondences”. *Proc. IEEE Conf. Computer Vision and Pattern Recognition*. 10. köt. IEEE Computer Soc. Press Washington DC. 1986, 112–115. old.

[52] Olivier D Faugeras és Martial Hebert. „A 3-D recognition and positioning algorithm using geometrical matching between primitive surfaces”. *Proceedings of the Eighth international joint conference on Artificial intelligence-Volume 2*. 1983, 996–1002. old.

[53] Shinji Umeyama. „Least-squares estimation of transformation parameters between two point patterns”. *IEEE Transactions on Pattern Analysis & Machine Intelligence* 13.04 (1991), 376–380. old.

[54] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige és Wolfram Burgard. „G2o: A general framework for graph optimization”. 2011. jún., 3607–3613. old. DOI: 10.1109/ICRA.2011.5979949.

[55] Manolis Lourakis és Antonis Argyros. „SBA: A Software Package for Generic Sparse Bundle Adjustment”. *ACM Trans. Math. Softw.* 36 (2009. jan.).

[56] Huang A.S. és et al. „Visual Odometry and Mapping for Autonomous Flight Using an RGB-D Camera”. *In: Christensen H., Khatib O. (eds) Robotics Research. Springer Tracts in Advanced Robotics* (2011).

[57] Bo Yuan és Yanduo Zhang. „A 3D Map Reconstruction Algorithm in Indoor Environment Based on RGB-D Information”. 2016. jan., 358–363. old. DOI: 10.1109/ISPDC.2016.59.

[58] Siciliano B. és Khatib O. „Springer-Verlag Berlin Heidelberg, Part C, RangeSensors”. *Springer Handbook of Robotics* (2008).

[59] R. C. Gonzalez és R. E. Woods. „Prentice-Hall”. *Digital Image Processing* (2002).

[60] Tai S. Lee. „Canny Edge Detection”. *Computer Vision* (2002), 15–385. old.

[61] Daniel Herrera C., Juho Kannala és Janne Heikkila. „Accurate and Practical Calibration of a Depth and Color Camera Pair”. 2011. aug., 437–445. old. ISBN: 978-3-642-23677-8. DOI: 10.1007/978-3-642-23678-5_52.

[62] Hanafiah Yussof. „Robot Localization and Map Building”. *In-Teh* (2010).

[63] J. Sturm, N. Engelhard, F. Endres, W. Burgard és D. Cremers. „A Benchmark for the Evaluation of RGB-D SLAM Systems”. *Proc. of the International Conference on Intelligent Robot Systems (IROS)*. 2012. okt. URL: https://cvg.cit.tum.de/data/datasets/rgbd-dataset/download (elérés dátuma 2023. 04. 12.).

[64] David G. Lowe. „Distinctive image features from scale-invariant keypoints”. *International Journal of Computer Vision* (2004), 9–110. old.

[65] Herbert Bay és Andreas Ess. „Speeded-UpRobustFeatures (SURF)". *Computer Vision and Image Understanding, Vol. 110, Issue 3* (2008), 346–359. old.

[66] Herbert Bay, Tinne Tuytelaars és Luc Van Gool. „SURF: Speeded up robust features". 3951. köt. 2006. júl., 404–417. old. ISBN: 978-3-540-33832-1. DOI: 10.1007/11744023_32.

[67] E. Rublee, Rabaud V., Konolige K. és Bradski G. „ORB: an efficient alternative to SIFT or SURF". *IEEE Int. Conf. on Computer Vision (ICCV)* (2011), 2564–2571. old.

[68] Luo Juan és Oubong Gwun. „A Comparison of SIFT, PCA-SIFT and SURF". *International Journal of Image Processing (IJIP)* (2010), 143–152. old.

[69] Emgu Corporation. *EMGU CV*. 2023. URL: https://www.emgu.com/wiki/index.php/Documentation (elérés dátuma 2023. 04. 12.).

[70] Jürgen Sturm, Stéphane Magnenat, Nikolas Engelhard, François Pomerleau, Francis Colas, Daniel Cremers, Roland Siegwart és Wolfram Burgard. „Towards a benchmark for RGB-D SLAM evaluation". *Proc. of the RGB-D Workshop on Advanced Reasoning with Depth Cameras at Robotics: Science and Systems Conf.(RSS)* (2011. jan.).

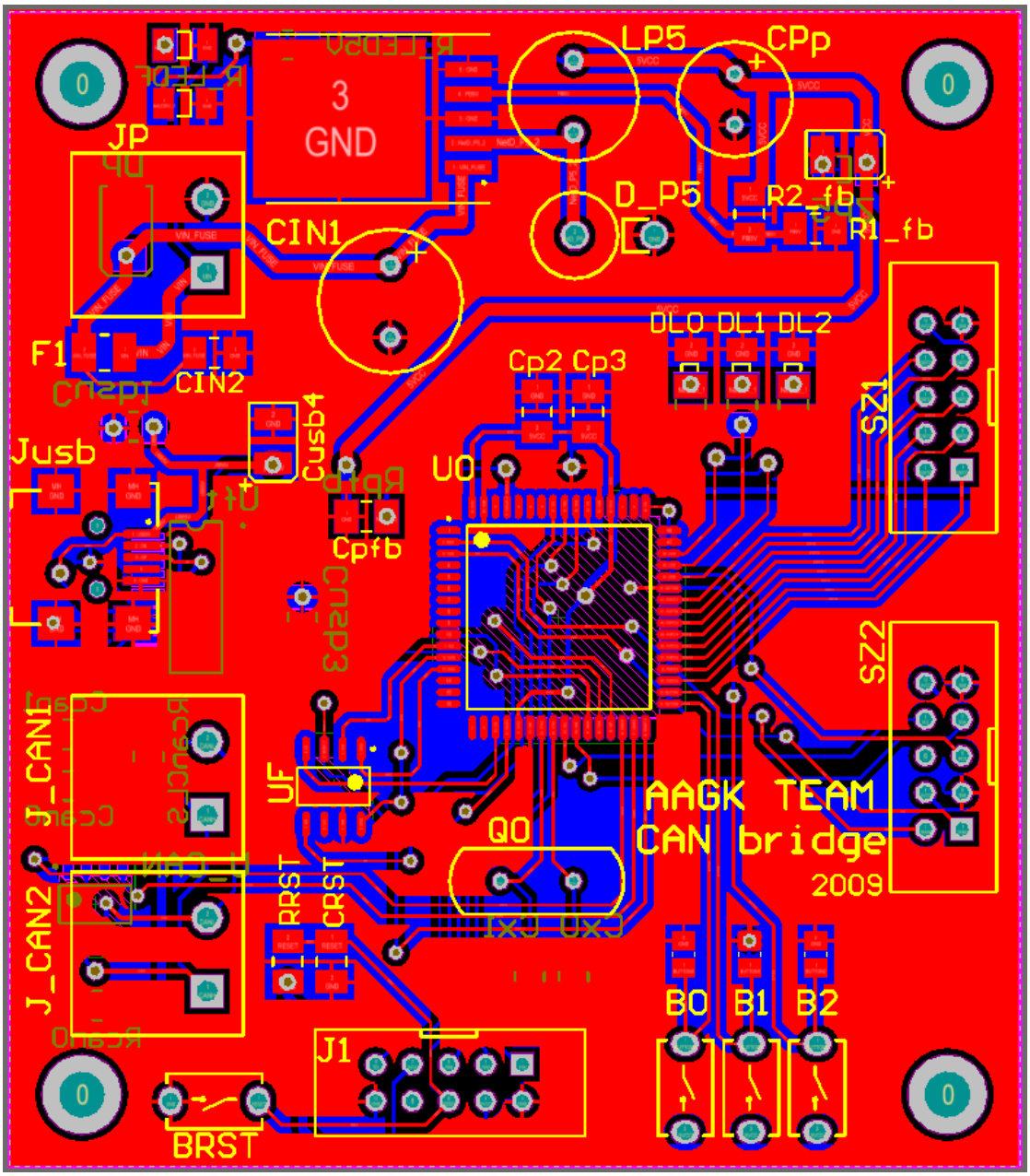
[71] Marek Kraft, Michał Nowicki, Rudi Penne, Adam Schmidt és Piotr Skrzypczyński. „Efficient RGB–D data processing for feature–based self–localization of mobile robots". *International Journal of Applied Mathematics and Computer Science* 26 (2016. márc.). DOI: 10.1515/amcs-2016-0005.

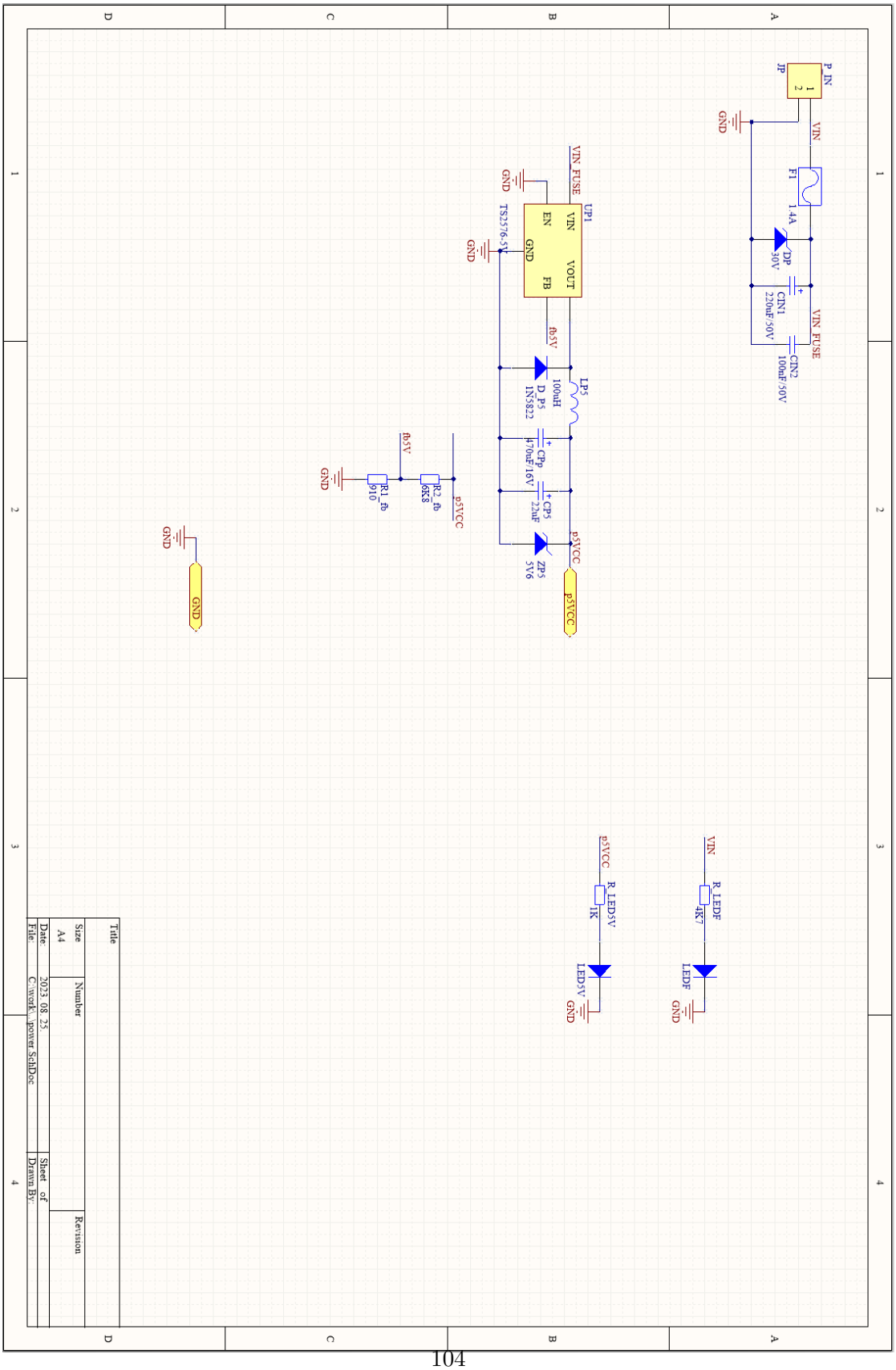
[72] Marek Kraft, Michał Nowicki, Adam Schmidt, Michał Fularz és Piotr Skrzypczyński. „Toward evaluation of visual navigation algorithms on RGB-D data from the first- and second-generation Kinect:" *Machine Vision and Applications* 28 (2017. febr.). DOI: 10.1007/s00138-016-0802-6.

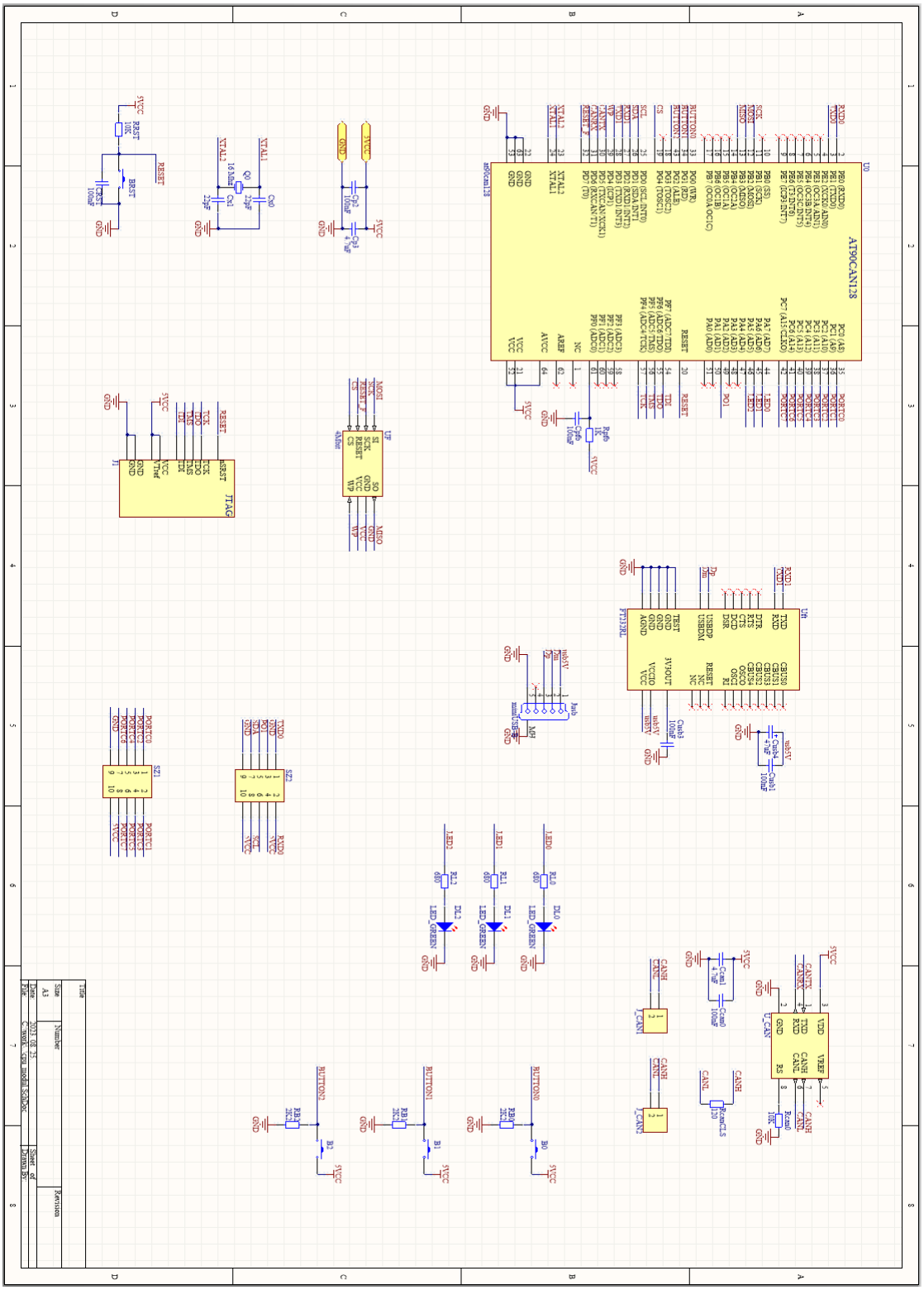
[73] Oguzhan Guclu és Ahmet Can. „Fast and effective loop closure detection to improve SLAM performance". *J. Intell. Robot. Syst.* (2017), 1–23. old.

Mellékletek

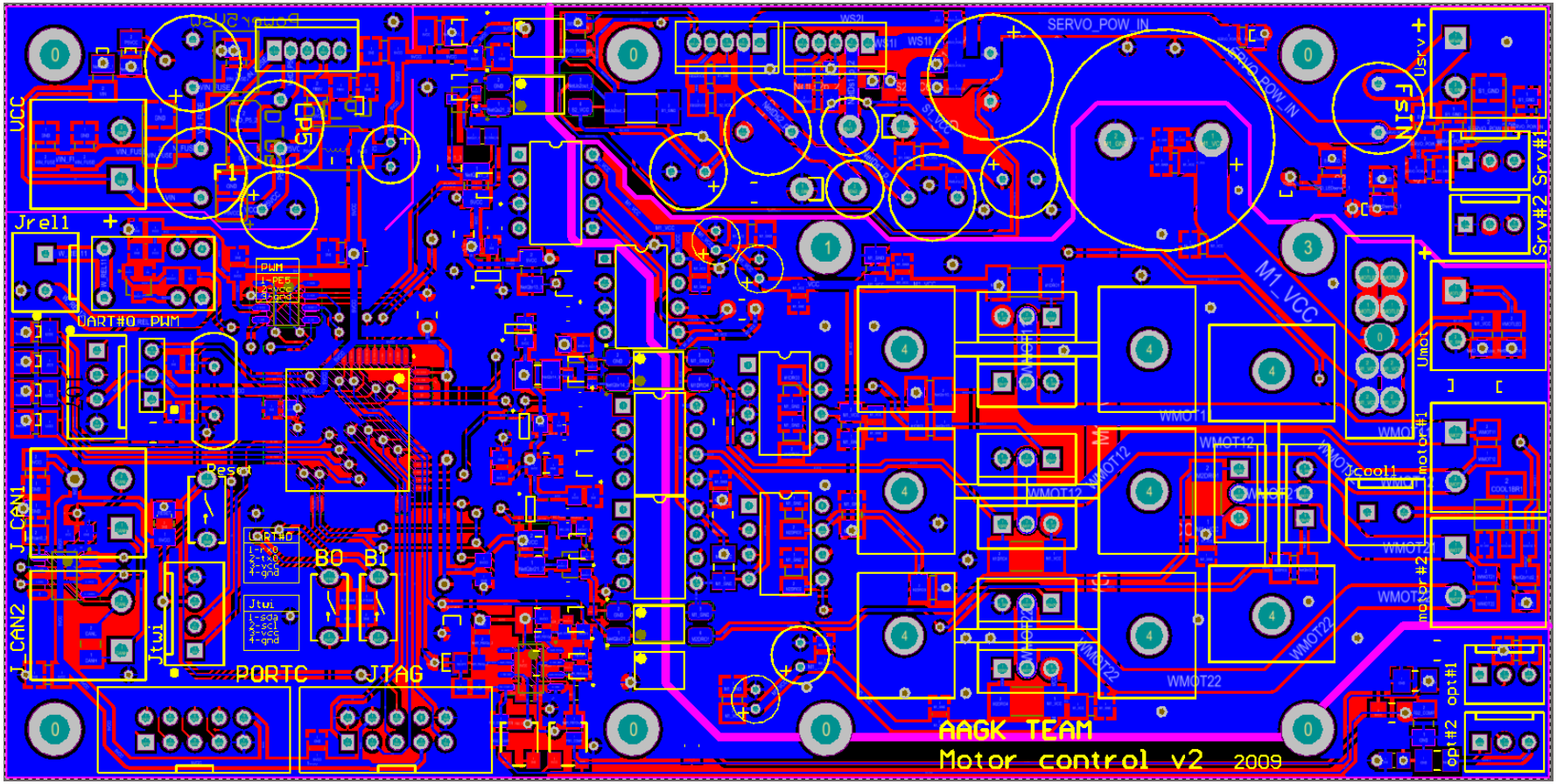
CAN busz átjáró PCB és kapcsolási rajza

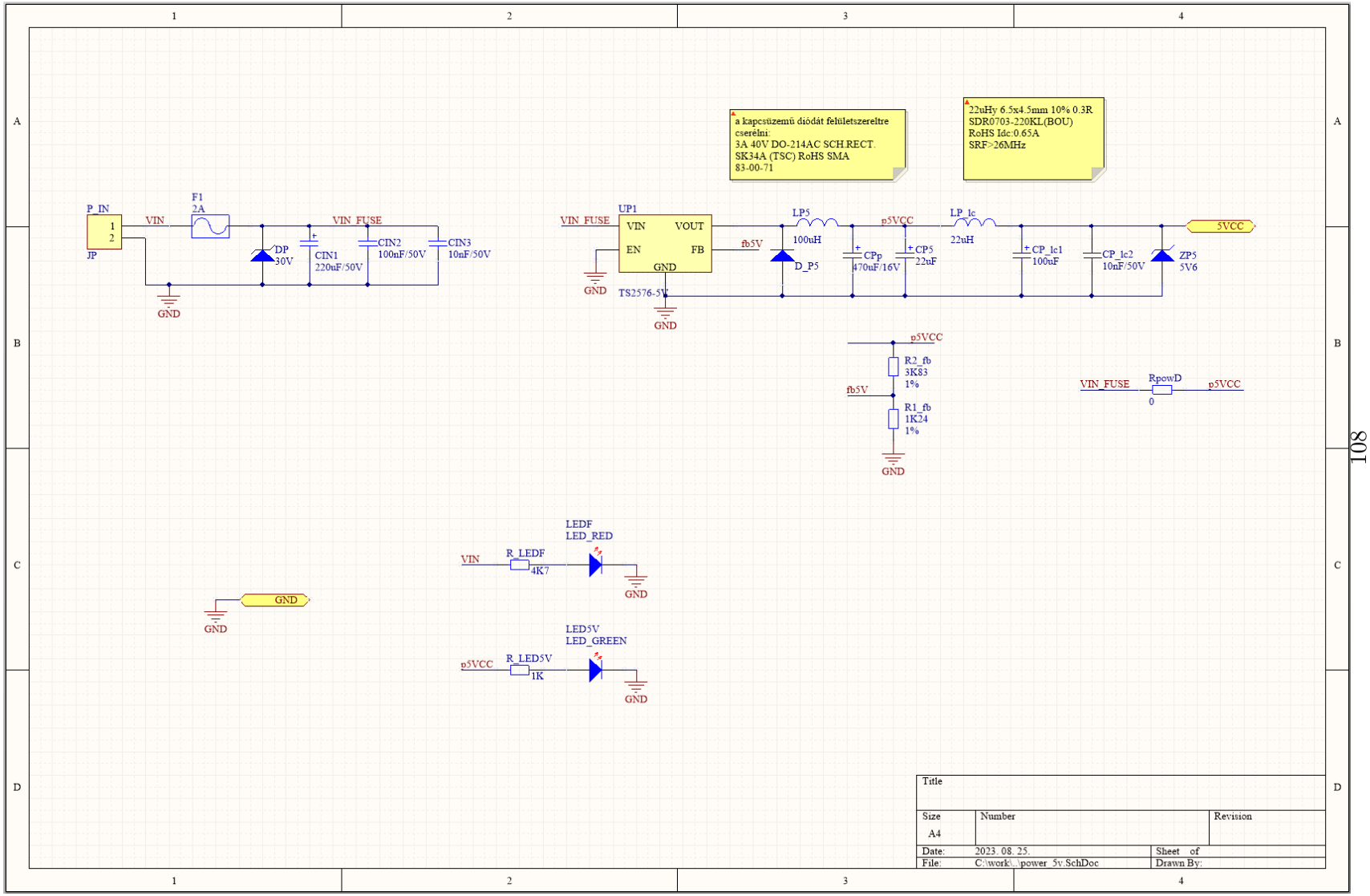

LP5
CPp
3
GND
JP
D_P5
R2_fb
R1_fb
CIN1
DL0 DL1 DL2
F1
CIN2
Cp2 Cp3
SZ1
Jusb
Cusb4
U0
Cpfb
SZ2
J_CAN1
UF
AAGK TEAM
CAN bridge
2009
Q0
J_CAN2
RRST
CRST
B0 B1 B2
J1
BRST

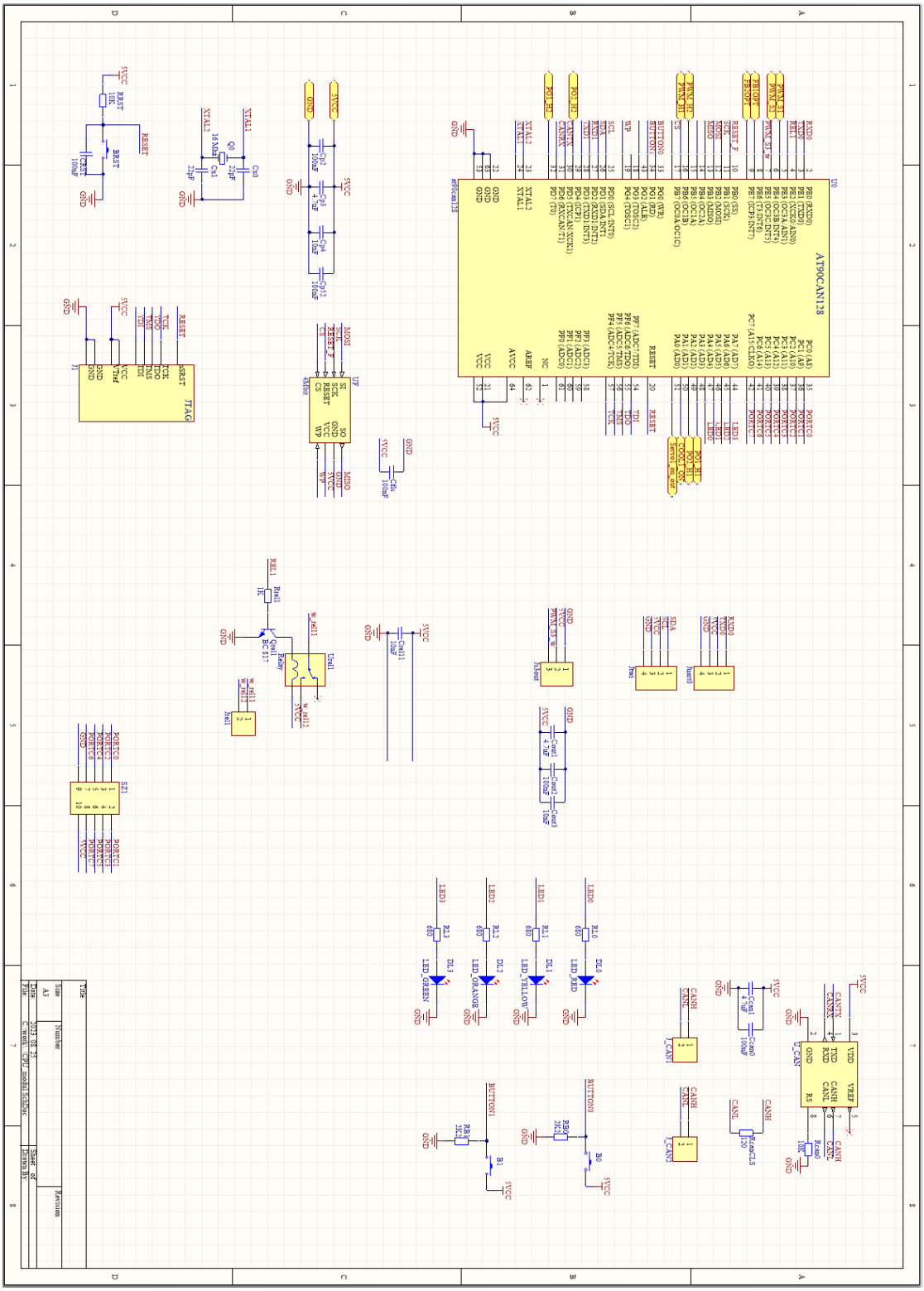

P_IN
JP
1
2
VIN
GND
F1
1.4A
ZDP
30V
CIN1
220uF/50V
CIN2
100nF/50V
VIN_FUSE
UP1
VIN_FUSE
VIN
EN
GND
VOUT
FB
TS2576-5V
fb5V
LP5
100uH
D_P5
1N5822
CP7
470uF/16V
CP5
22uF
ZP5
5V6
p5VCC
R1_fb
910
R2_fb
6K8
R_LEDF
4K7
LEDF
R_LED5V
1K
LED5V
Title
Size
A4
Number
Revision
Date
2023-08-25
Sheet of
File
C:\work\...\power.SchDoc
Drawn By:

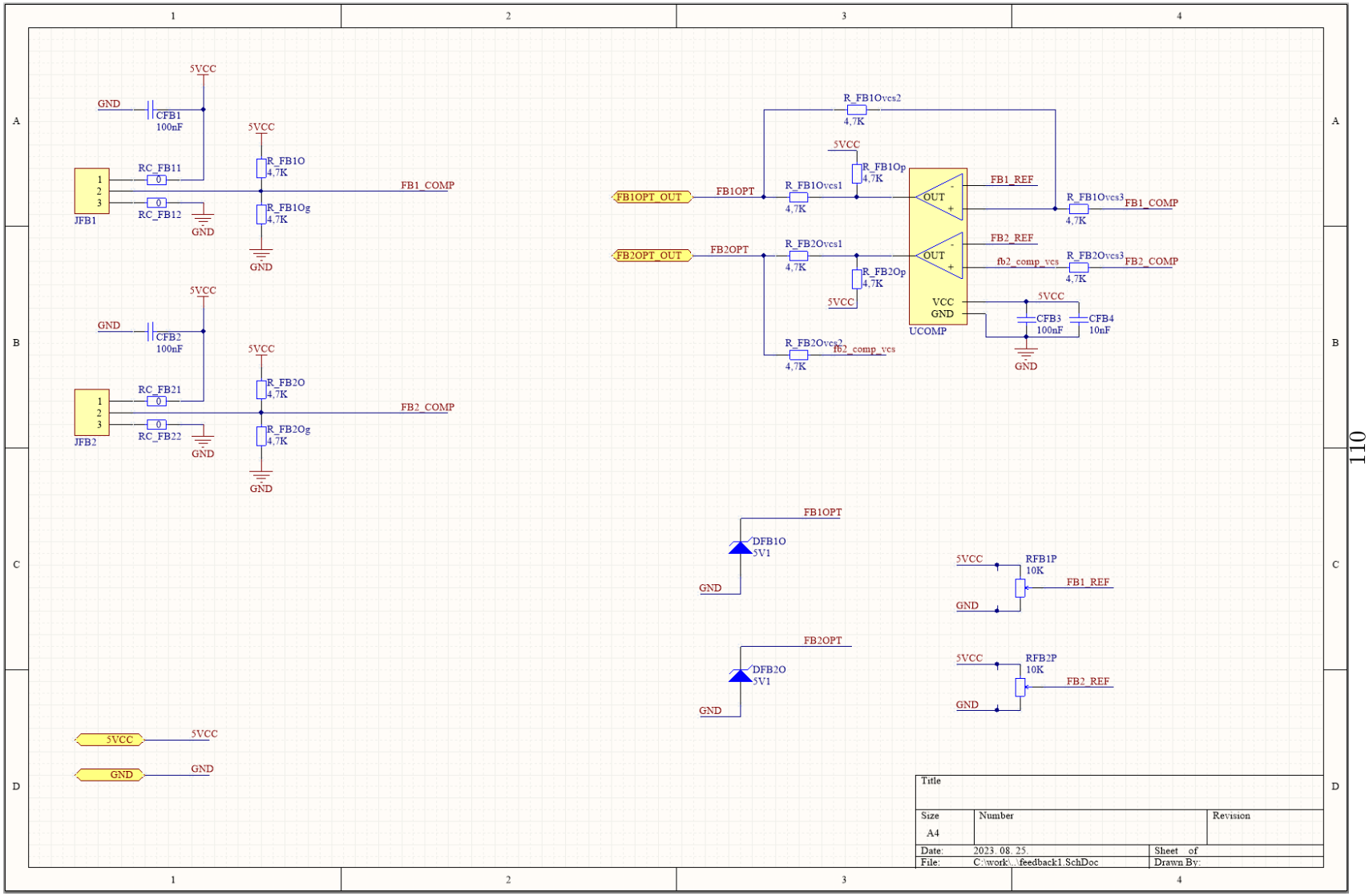


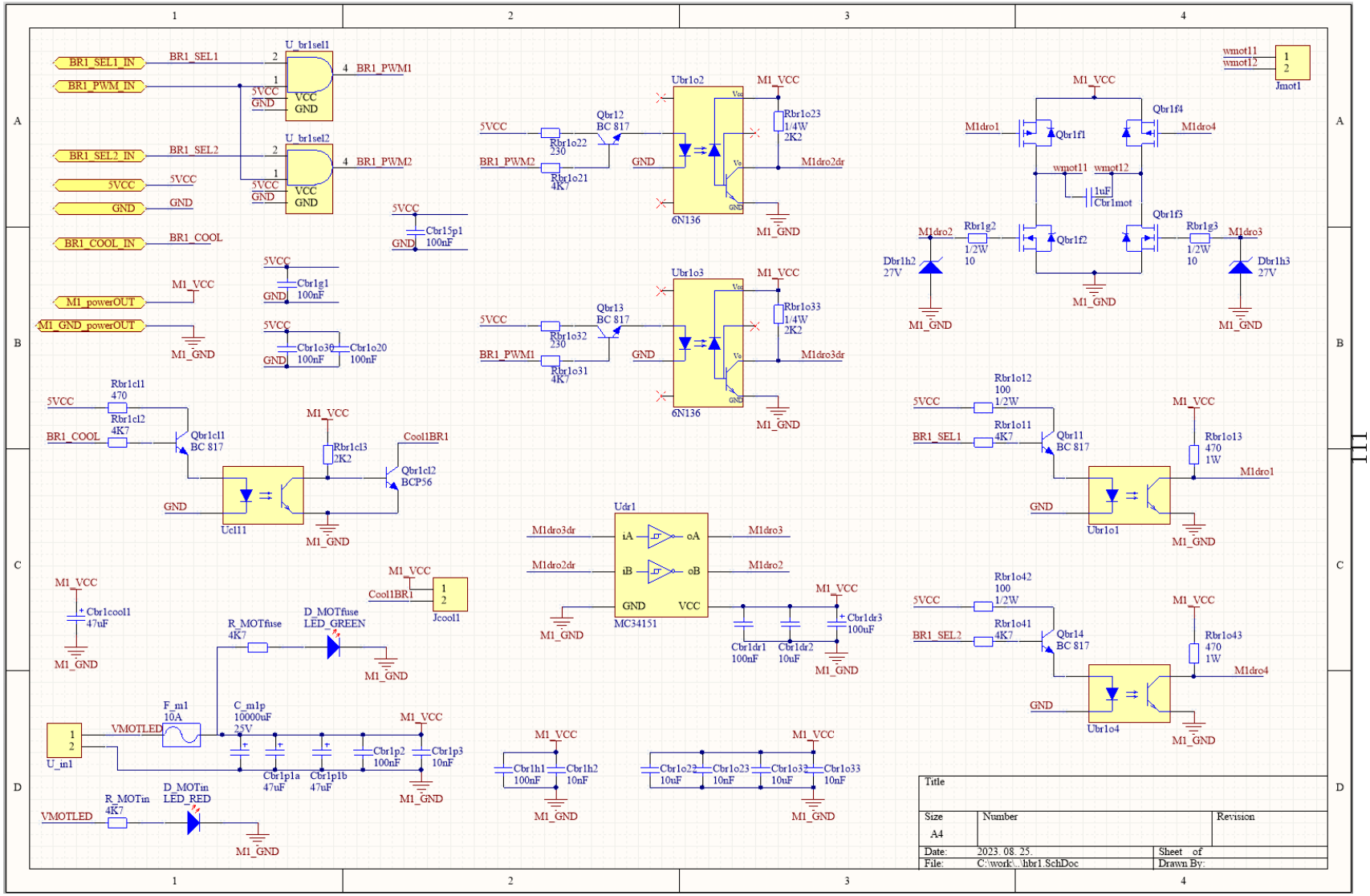
Motormeghajtó modul PCB és kapcsolási rajza

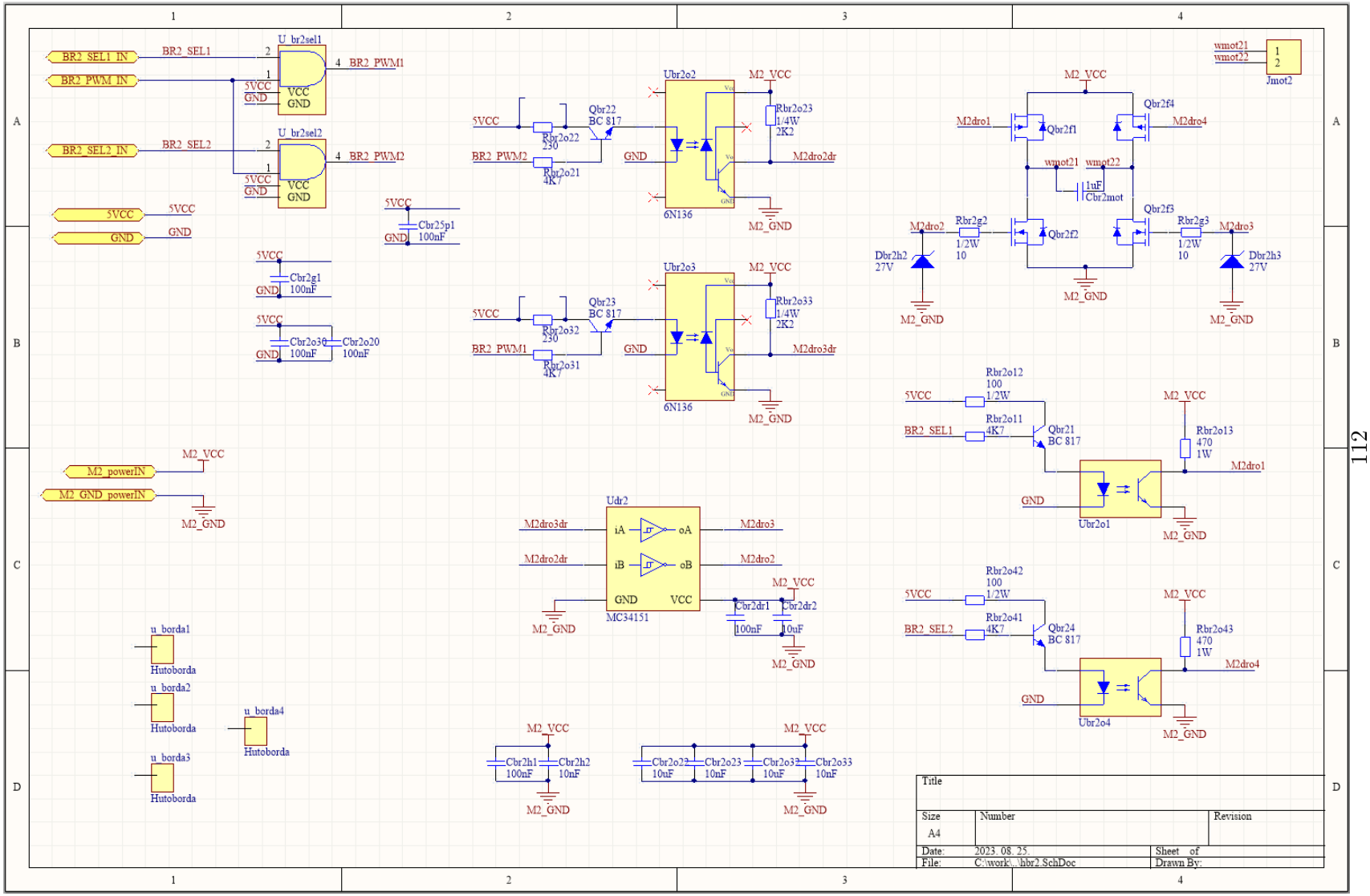

SERVO_POW_IN
UCC
Jrel1
F1
PWM
UART#0 PWM
J_CAN1
J_CAN2
Res
UART#0
B0
B1
Jtwi
PORTC
JTAG
M1_VCC
WMOT11
WMOT12
WMOT21
WMOT22
Srv#2 Srv#1
motor#1
motor#2
opt#2 opt#1
AAGK TEAM
Motor control v2 2009

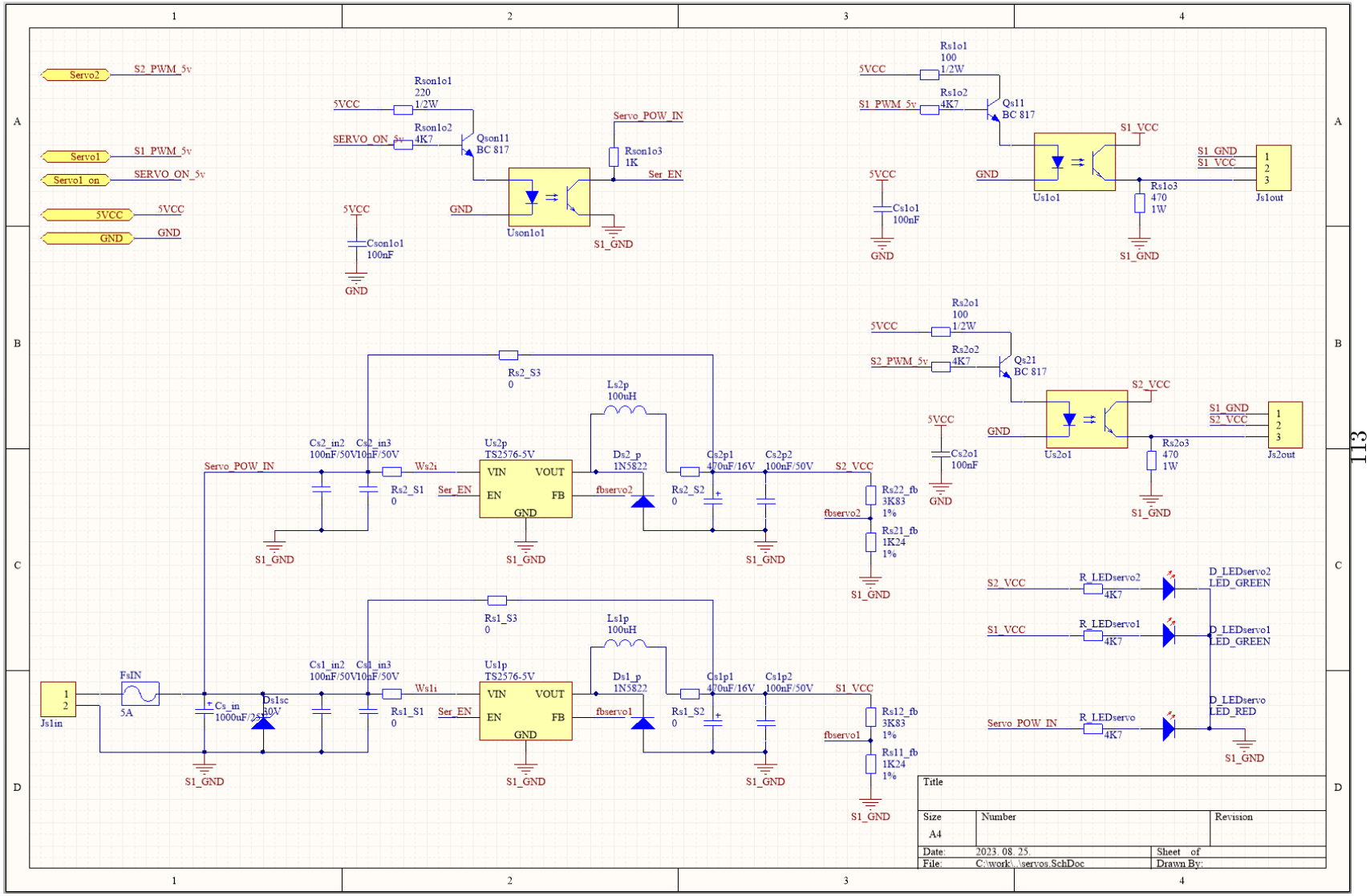

a kapcsüzemü diódát felületszereltre cserélni:
3A 40V DO-214AC SCH.RECT.
SK34A (TSC) RoHS SMA
83-00-71
22uHy 6.5x4.5mm 10% 0.3R
SDR0703-220KL(BOU)
RoHS Idc:0.65A
SRF>26MHz
P_IN
JP
VIN
F1
2A
VIN_FUSE
DP
30V
CIN1
220uF/50V
CIN2
100nF/50V
CIN3
10nF/50V
GND
UP1
VIN
VOUT
EN
FB
GND
TS2576-5V
fb5V
LP5
100uH
D_P5
p5VCC
CPp
470uF/16V
CP5
22uF
LP_lc
22uH
CP_lc1
100uF
CP_lc2
10nF/50V
ZP5
5V6
5VCC
R2_fb
3K83
1%
R1_fb
1K24
1%
RpowD
0
LEDF
LED_RED
R_LEDF
4K7
LED5V
LED_GREEN
R_LED5V
1K
Title
Size
A4
Number
Revision
Date: 2023. 08. 25.
File: C:\work\...\power_5v.SchDoc
Sheet of
Drawn By:




5VCC
GND
CFB1
100nF
RC_FB11
0
RC_FB12
JFB1
R_FB1O
4,7K
R_FB1Og
FB1_COMP
CFB2
RC_FB21
RC_FB22
JFB2
R_FB2O
R_FB2Og
FB2_COMP
R_FB1Ovcs2
R_FB1Op
FB1OPT_OUT
FB1OPT
R_FB1Ovcs1
FB1_REF
R_FB1Ovcs3
FB2OPT_OUT
FB2OPT
R_FB2Ovcs1
R_FB2Op
FB2_REF
fb2_comp_vcs
R_FB2Ovcs3
R_FB2Ovcs2
UCOMP
VCC
CFB3
100nF
CFB4
10nF
DFB1O
5V1
DFB2O
RFB1P
10K
RFB2P
Title
Size
A4
Number
Revision
Date:
2023. 08. 25.
Sheet of
File:
C:\work\...\feedback1.SchDoc
Drawn By:


BR1_SEL1_IN
BR1_PWM_IN
BR1_SEL2_IN
5VCC
GND
BR1_COOL_IN
M1_powerOUT
M1_GND_powerOUT
U_br1sel1
U_br1sel2
BR1_SEL1
BR1_SEL2
BR1_COOL
BR1_PWM1
BR1_PWM2
Cbr15p1
Cbr1g1
Cbr1o30
Cbr1o20
100nF
Rbr1cl1
Rbr1cl2
Qbr1cl1
BC 817
Qbr1cl2
BCP56
Rbr1cl3
2K2
Ucl11
Cool1BR1
Ubr1o2
Ubr1o3
Qbr12
Qbr13
Rbr1o22
Rbr1o32
Rbr1o21
Rbr1o31
230
4K7
6N136
Rbr1o23
Rbr1o33
1/4W
M1dro2dr
M1dro3dr
Udr1
MC34151
iA
iB
oA
oB
M1dro3
M1dro2
Cbr1dr1
Cbr1dr2
Cbr1dr3
100uF
10nF
wmot11
wmot12
Jmot1
M1dro1
M1dro4
Qbr1f1
Qbr1f4
Qbr1f2
Qbr1f3
1uF
Cbr1mot
Rbr1g2
Rbr1g3
1/2W
10
Dbr1h2
Dbr1h3
27V
Rbr1o12
Rbr1o42
100
Rbr1o11
Rbr1o41
Qbr11
Qbr14
Rbr1o13
Rbr1o43
1W
Ubr1o1
Ubr1o4
Cbr1cool1
47uF
Jcool1
R_MOTfuse
D_MOTfuse
LED_GREEN
F_m1
10A
C_m1p
10000uF
25V
VMOTLED
U_in1
Cbr1p1a
Cbr1p1b
Cbr1p2
Cbr1p3
R_MOTin
D_MOTin
LED_RED
Cbr1h1
Cbr1h2
Cbr1o22
Cbr1o23
Cbr1o32
Cbr1o33
Title
Size
A4
Number
Revision
Date:
2023. 08. 25.
Sheet of
File:
C:\work\...\hbr1.SchDoc
Drawn By:


BR2_SEL1_IN
BR2_PWM_IN
BR2_SEL2_IN
5VCC
GND
M2_powerIN
M2_GND_powerIN
U_br2sel1
U_br2sel2
BR2_SEL1
BR2_SEL2
BR2_PWM1
BR2_PWM2
Cbr25p1 100nF
Cbr2g1 100nF
Cbr2o36 100nF
Cbr2o20 100nF
Ubr2o2
Ubr2o3
Qbr22 BC 817
Qbr23 BC 817
Rbr2o22 230
Rbr2o21 4K7
Rbr2o32 230
Rbr2o31 4K7
Rbr2o23 1/4W 2K2
Rbr2o33 1/4W 2K2
6N136
M2dro2dr
M2dro3dr
Udr2
MC34151
Cbr2dr1 100nF
Cbr2dr2 10uF
Cbr2h1 100nF
Cbr2h2 10nF
Cbr2o22 10nF
Cbr2o23 10nF
Cbr2o32 10nF
Cbr2o33 10nF
u_borda1
u_borda2
u_borda3
u_borda4
Hutoborda
wmot21
wmot22
Jmot2
Qbr2f1
Qbr2f2
Qbr2f3
Qbr2f4
M2dro1
M2dro2
M2dro3
M2dro4
Rbr2g2 1/2W 10
Rbr2g3 1/2W 10
Dbr2h2 27V
Dbr2h3 27V
Cbr2mot 1uF
Rbr2o12 100 1/2W
Rbr2o11 4K7
Qbr21 BC 817
Rbr2o13 470 1W
Ubr2o1
Rbr2o42 100 1/2W
Rbr2o41 4K7
Qbr24 BC 817
Rbr2o43 470 1W
Ubr2o4
Title
Size A4
Number
Revision
Date: 2023. 08. 25.
Sheet of
File: C:\work\...\hbr2.SchDoc
Drawn By:


Servo2
S2_PWM_5v
Servo1
S1_PWM_5v
Servo1_on
SERVO_ON_5v
5VCC
GND
Rson1o1
220
1/2W
Rson1o2
4K7
Qson11
BC 817
Servo_POW_IN
Rson1o3
1K
Ser_EN
Cson1o1
100nF
Uson1o1
S1_GND
Rs1o1
100
1/2W
Rs1o2
4K7
Qs11
BC 817
S1_VCC
Us1o1
Rs1o3
470
1W
Js1out
Cs1o1
100nF
Rs2o1
100
1/2W
Rs2o2
4K7
Qs21
BC 817
S2_VCC
Us2o1
Rs2o3
470
1W
Js2out
Cs2o1
100nF
Rs2_S3
0
Ls2p
100uH
Cs2_in2
100nF/50V
Cs2_in3
10nF/50V
Us2p
TS2576-5V
VIN
VOUT
EN
FB
GND
Ws2i
Rs2_S1
0
Ds2_p
1N5822
Cs2p1
470uF/16V
Cs2p2
100nF/50V
Rs2_S2
0
Rs22_fb
3K83
1%
Rs21_fb
1K24
1%
fbservo2
Rs1_S3
0
Ls1p
100uH
Cs1_in2
100nF/50V
Cs1_in3
10nF/50V
Us1p
TS2576-5V
Ws1i
Rs1_S1
0
Ds1_p
1N5822
Cs1p1
470uF/16V
Cs1p2
100nF/50V
Rs1_S2
0
Rs12_fb
3K83
1%
Rs11_fb
1K24
1%
fbservo1
FsIN
5A
Js1in
Cs_in
1000uF/25V
Ds1sc
30V
R_LEDservo2
R_LEDservo1
R_LEDservo
4K7
D_LEDservo2
LED_GREEN
D_LEDservo1
LED_GREEN
D_LEDservo
LED_RED
Title
Size
A4
Number
Revision
Date:
2023. 08. 25.
Sheet of
File:
C:\work\...\servos.SchDoc
Drawn By: