ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

DIPLOMAMUNKA

OE-NIK Hallgató neve: **Illés-Tomka András**

2022 Hallgató törzskönyvi száma: **T/005375/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Kiberfizikai Rendszerek Intézet

DIPLOMAMUNKA FELADATLAP

Hallgató neve: **Illés-Tomka András**
Törzskönyvi száma: T/005375/FI12904/N
Neptun kódja: [redacted]

A diplomamunka címe:

Kinect szenzoron alapuló három dimenziós térképépítő rendszer
Three dimensional map building system based on Kinect sensor

Intézményi konzulens: Somlyai László
Külső konzulens:

Beadási határidő: 2022. május 15.

A feladat

A diplomamunka célja Kinect szenzor segítségével történő három dimenziós térképépítő eljárás megvalósítása. Elsődlegesen a Kinect szenzor kamerájának segítéségével legyen képes a rendszer a lokális és globális környezetének megismerésére, rögzítésére, a tárolt és feldolgozott adatokból pedig állítson elő háromdimenziós térképet. A dolgozatot a szoftverfejlesztés alaptevékenységeinek megfelelően dolgozza ki és adja közre.

A diplomamunkának tartalmaznia kell:

- Tekintse át az RGBD szenzorok működési elvét és azok adatainak feldolgozásra irányuló módszereket,
- A szakirodalom megismerése, a képjellemzők azonosítását lehetővé tevő algoritmusokról,
- Irodalomkutatás segítségével mutassa be a lokalizációs és navigációs eljárások jellemzőit, valamint a térkép reprezentációk lehetőségeit,
- Elemezze a lehetséges megvalósítási alternatívákat,
- Valósítsa meg alkalmazását,
- a megvalósítás menetét,
- Ismertesse a fejlesztés menetét, döntéseinek indoklását valamint a tesztelés tervét és eredményét,
- Vázolja a továbbfejlesztési lehetőségeket.

ÓBUDAI EGYETEM * Neumann János Informatikai Kar * 1. Budapest

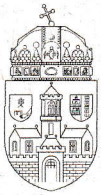
.....
Dr. Fleiner Rita
intézetigazgató

A diplomamunka elévülésének határideje: **2024. május 15.**
(OE TVSz 55.§ szerint)

A diplomamunkát beadásra alkalmasnak tartom:

.....
külső konzulens

.....
intézményi konzulens



ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a diplomamunka saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült diplomamunkámban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja.

Budapest, 2022. május 15.

.....

hallgató aláírása

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar
9. sz. melléklet

KONZULTÁCIÓS NAPLÓ

Hallgató neve: **Illés-Tomka András**

Neptun kód: [redacted]

Tagozat: **Esti**

Telefon: [redacted]

Levelezési cím (pl: lakcím): [redacted]

Szakdolgozat / **Diplomamunka**[1] címe magyarul:

Kinect szenzoron alapuló három dimenziós térképépítő rendszer

Szakdolgozat / **Diplomamunka**[2] címe angolul:

Three dimensional map building system based on Kinect sensor

Intézményi konzulens: **Somlyai László**

Külső konzulens:

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2020-04-09	Lokalizációs eljárások	Somlyi László
2.	2020-04-23	Vizuális odometria	Somlyi László
3.	2020-05-07	SLAM	Somlyi László
4.	2020-05-14	Rendszerterv / javítások	Somlyi László

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / **Diplomamunka 2** / Diplomamunka 3 / Diplomamunka 4[3] tantárgy követelményét teljesítette, **beszámolóra** / védésre[4] bocsátható.

Budapest, 2020.05.21.

Somlyi László
intézményi konzulens

1 Megfelelő aláhúzandó!
2 Megfelelő aláhúzandó!
3 Megfelelő aláhúzandó!
4 Megfelelő aláhúzandó!

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar
9. sz. melléklet

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Neptun kód: Tagozat:

Illés-Tomka András..... [redacted] ..... **Esti**.....

Telefon: Levelezési cím (pl: lakcím):

[redacted].....

Szakdolgozat / **Diplomamunka**[1] címe magyarul:

Kinect szenzoron alapuló három dimenziós térképépítő rendszer

Szakdolgozat / **Diplomamunka**[2] címe angolul:

Three dimensional map building system based on Kinect sensor

Intézményi konzulens: Külső konzulens:

Somlyai László..... .....

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2021-03-29	Diplomamunk 2 eredményei / észrevételek / javaslatok, VO javítás	Somlyai László
2.	2021-04-12	Feature detectors	Somlyai László
3.	2021-04-26	SLAM + VSLAM	Somlyai László
4.	2021-05-10	ROS / Rendszerterv / Javítások	Somlyai László

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplomamunka 2 / **Diplomamunka 3** / Diplomamunka 4[3] tantárgy követelményét teljesítette, **beszámolóra** / védésre[4] bocsátható.

Budapest, 2021.05.14.

Somlyai László
.....
intézményi konzulens

1 Megfelelő aláhúzandó!
2 Megfelelő aláhúzandó!
3 Megfelelő aláhúzandó!
4 Megfelelő aláhúzandó!

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar
9. sz. melléklet

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Neptun kód: Tagozat:

Illés-Tomka András..... [redacted] ..... **Esti**.....

Telefon: Levelezési cím (pl: lakcím):

[redacted].....

Szakdolgozat / **Diplomamunka**[1] címe magyarul:

Kinect szenzoron alapuló három dimenziós térképépítő rendszer

Szakdolgozat / **Diplomamunka**[2] címe angolul:

Three dimensional map building system based on Kinect sensor

Intézményi konzulens: Külső konzulens:

Somlyai László..... .....

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2022-03-04	Pontfelhő konstruálás	Somlyai László
2.	2022-03-25	PyQT és frontend fejlesztés	Somlyai László
3.	2022-04-13	Vizuális odometria + slam funkciók implementálása	Somlyai László
4.	2022-05-06	Rendszerterv finomítások	Somlyai László

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplomamunka 2 / Diplomamunka 3 / **Diplomamunka 4**[3] tantárgy követelményét teljesítette, beszámolóra / **védésre**[4] bocsátható.

Budapest, 2022.05.13.

.....Somlyai László.....
intézményi konzulens

1 Megfelelő aláhúzandó!
2 Megfelelő aláhúzandó!
3 Megfelelő aláhúzandó!
4 Megfelelő aláhúzandó!

Megköszönöm

SOMLYAI LÁSZLÓ,

az Óbudai Egyetem Neumann János Informatikai Kar, Kiberfizikai Rendszerek Intézet tanszéki mérnökének

segítségét, aki a diplomamunkám megírása során felvetődő valamennyi probléma megoldásában a legmesszebbmenőkig segítségemre volt.

Tartalomjegyzék

A Feladat **12**

1. Bevezető **13**

2. Az autonóm navigáció és lokalizáció jellemzői, nehézségei **14**

2.1. Lokalizáció fajtái és megoldásai 15

2.1.1. Odometria 16

2.1.2. Vizuális odometria 16

2.1.3. GPS 17

2.2. Lokalizációs megoldások összefoglalása 18

3. A vizuális odometria alapjai és eszközei **20**

3.1. A vizuális odometria fogalma, építőelemei és elterjedése 20

3.1.1. A kameratípusok 20

3.1.2. A vizuális odometria algoritmusa 25

4. SLAM **38**

4.1. SLAM eljárás alapjai 39

4.2. Matematikai leírás 40

4.2.1. Markov-lánc 41

4.3. Térkép fajták 42

4.3.1. Metrikus térképek 42

4.3.2. Topológikus térképek 43

4.4. SLAM kategóriák 43

4.5. Vizuális-SLAM 44

5. Rendszerterv **46**

6. Felhasznált szoftverek, gyűjtemények bemutatása **48**

6.1. Robot Operating System 48

6.1.1. ROS szerkezet 49

6.1.2. ROS kommunikációs típsuok 49
6.1.3. Összefoglalás 49
6.2. Libfreenect2 50
6.3. iai_kinect package 50
6.4. OpenCV 2.4 51
6.5. PyQt 51
6.6. Point Cloud Library 52

7. Tervezés és fejlesztés dokumentálása 54
7.1. Infrastruktúra 54
7.2. Kalibráció 54
7.2.1. Kalibráció menete 55
7.2.2. Nehézségek 55
7.3. Felhasználói kezelőfelület 57
7.3.1. UI 57
7.3.2. Funkciók 57
7.4. Architektúra 58
7.4.1. Frontend 59
7.4.2. Backend 60
7.5. Implementáció részletei 61
7.5.1. Osztályok és funkcióik 61
7.5.2. Eljárás részei 62

8. Eredmények 67

9. Továbbfejlesztési lehetőségek 68
9.1. SLAM eljárás pontosítása 68
9.2. Keretrendszer és könyvátrcsomagok frissítése 68
9.3. Szabályozó integrálása 69

Összefoglalás 70

Summary **71**

Ábrajegyzék **72**

Táblajegyzék **73**

Irodalomjegyzék **74**

Mellékletek **78**

A. Környezet installálása **78**

A.1. ROS Indigo telepítése 78

A.2. Libfreenect2 driver installálása 80

A.3. iai_kinect ROS package telepítése 81

A.4. VTK, PCL és PyQT telepítése 82

A feladat

Diplomamunkám feladataként egy három dimenziós (3D) térképépítő eljárás lefejlesztését tűztem ki célul. Ezt a témát azért választottam, mert bár a mindennapi munkámtól távol áll, a robotika területéhez kiemelten kapcsolódik és széleskörű felhasználási lehetőséget nyújt az alkalmazása.

A térképépítő megoldások többféle kategória szerint csoportosíthatóak. Megoldásomat a vizuális szenzorokra építő eljárásokra alapoztam . Eszközként a Microsoft Kinect v2 RGB-D szenzorát választottam, az implementációhoz pedig a robotika világában standard keretrendszernek számító ROS-t használtam.

Bár nem volt jártasságom és megfelelő kompetenciám e témakörben, a feladat megvalósításához igyekeztem szerteágazó tudásra szert tenni, a szakirodalmat részletesen áttekinteni és annak vonatkozó részeit diplomamunkámban is fókuszba állítani.

A feladat részeként áttekintésre került:

- a lokalizációs eljárások lehetőségei,
- a vizuális odometria és SLAM elméleti háttere,
- a hardver megoldások széles skálája,
- és az elérhető megoldások halmaza.

Mindezek alapján megszületett a fejlesztendő rendszer

- terve és architektúrája,
- a lefejlesztett prototípusa,
- valamint az eredmények kiértékelése.

A környezet telepítése, a tervezés részletezése, valamint az implementáció kulcspontjai dokumentálásra kerültek az újrafelhasználhatóságot figyelembe véve, így a diplomamunkám és mellékletei minden végrehajtott lépést tartalmaznak.

Az általam választott terület és feladat komplexitása számtalan kérdést vont maga után, melyekkel igyekeztem legjobb tudásom szerint diplomamunkában részletesen foglalkozni.

1. Bevezető

A háromdimenziós térképépítő eljárások a robotika kiemelt fontosságú részterületét képezik. Olyan területről van szó, amely többféle tudományterület vívmányait hasznosítja. Ahhoz, hogy ezek összefüggéseit, egymásra való hatásait megismerjük, bevezetőmben szeretném röviden áttekinteni a különböző tudományterületeket.

A gépi látás (Computer Vision) mint összetett tudományterület az utóbbi néhány évtizedben indult fejlődésnek. Kezdetben szigorúan a képek, videók analízisére összpontosított, azon belül is a képeken lévő tárgyak, objektumok felismerésére. Ma már ennél szerteágazóbb a tudományterület, célja a digitális képek tartalmának kontextusbeli megértése, a belőlük való hasznos információ kinyerése, reprezentálása, modellezése és felhasználása - hasonlóan az emberi látáshoz, feldolgozáshoz és tároláshoz.

A mesterséges intelligencia (Artifical Intelligence) – szoros összefüggésben a gépi látással – szintén az utóbbi években indult hatalmas fejlődésnek. Nem véletlen, hiszen a jelentősen kifinomult érzékelők, a megsokszorozódott számítási kapacitás, a gépi látásából eredő megannyi információ új távlatokat nyitott az ember géppel való együttműködésében.

A döntéstámogatás, az automatizálás, a környezetben történő változásokra való hatékonyabb – akár emberi beavatkozás nélküli – reagálás és a megismert helyzetekből való jövőbeli felismerés (tanulás) mind-mind részét képezik e szerteágazó világnak. A váratlan helyzetek, katasztrófák (legyen az emberi vagy természeti) és a katonai hadviselés mind-mind igénylik a hatékonyabb felderítést, gyorsabb zavarelhárításra vagy problémamegoldást, ami ma már AI és CV nélkül elképzelhetetlen lenne (pl.: Fukushima, 2011).

Mindezek alapján e két terület ma már egyértelműen része a hétköznapjainknak. Térhódítása jelentősen növekszik azokon a területeken is, ahol korábban elképzelhetetlen volt (ipar, mezőgazdaság, pénzügyi szektor. orvostudomány, közlekedés, szórakoztató ipar stb.). A robotika sem kivétel ezalól.

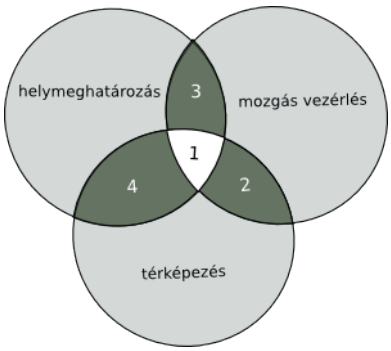
A robotika a gépi látás eredményeinek első számú felhasználója. A kollaboratív együttműködéshez, az automatizált munkavégzéshez nélkülözhetetlen a környezet jellemzőinek megismerése, a tárgyak azonosítása, a veszélyhelyzet felismerése. Diplomamunkámban e diszciplína egyik jelentős problémakörével, a lokalizációval és navigációval kívánok foglalkozni, valamint olyan megoldást fejleszteni, ami mindhárom terület eredményeit ötvözi.

2. Az autonóm navigáció és lokalizáció jellemzői, nehézségei

A mobil robotok egyik legnagyobb kihívása a mindennapi környezetünkben való teljes körű önálló, autonóm munkavégzés megvalósítása. Ez – a feladat végrehajtása mellett – a környezetnek megismerését (tárgyak észlelése, ütközések elkerülése, nem biztonságos körülmények felismerése), az abban való tájékozódást, navigációt, útkeresést, valamint az ehhez nélkülözhetetlen helymeghatározást (lokalizáció és orientáció) jelenti. Az utóbbi évek jelentős hardver- és szoftvertechnológiai innovációi (érzékelés, elosztott feldolgozás, gépi látás) nagyban hozzájárult a terület fejlődéséhez, segítségével egyre pontosabb megoldások, implementációk látnak napvilágot. A probléma komplexitása és diverzitása miatt a teljes áttörés azonban még várat magára.

Az autonóm navigáció – legyen az beltéri vagy kültéri – legfontosabb jellemzője, hogy a robot az elérni kívánt cél érdekében képes a saját szenzorai segítségével környezetét érzékelni, valamint abból adatot kinyerni (térkép), elmozdulását detektálni és környezetéhez viszonyítani (hely- és helyzetmeghatározás), az így kapott adatok alapján pedig mozgásvezérlő rendszerén keresztül aktuátorait szabályozni (mozgás tervezés, vezérlés). A robot tehát a trajektória tervezéséhez és végrehajtásához olyan összetett funkciókat végez, mint: érzékelés, feldolgozás, döntéshozás és -végrehajtás.[34]

Az 1. ábra a navigáció alkotóelemeit és feladatait mutatja a [38]-ben található ábra alapján: 1.integrált megoldás; 2.exploráció: környezet felfedezés optimális útvonal bejárás mellett; 3.aktív helymeghatározás: cél a pontos lokalizáció, ami érdekében cselekvését, útvonalát is optimalizálhatja; 4.SLAM: a jó térképhez helymeghatározás, a helymeghatározáshoz pedig térkép kell.



1. ábra. A helymeghatározás, az útvonal tervezés és a térképezés kapcsolata.

A lokalizáció, mely magába foglalja a robot elmozdulásának és orientációjának a meghatározását is, egy összetett probléma. Alapvető cél, hogy a végrehajtás során minden időpillanatban képesek legyünk a robot pozíciójának lehető legpontosabb meghatározására, azaz arra, hogy a robot „hol van?" és „merre néz?" [6]. Ezen az információnak hiánya negatív irányba befolyásolja a pályagörbe tervezését, a mozgás vezérlését, valamint a navigáció sikerességét is, mely fogalmak természetesen nem választhatók el egymástól.[38]

2.1. Lokalizáció fajtái és megoldásai

A bizonytalanságot csökkentő, pontos helymeghatározás elengedhetetlen a tökéletes trajektória tervezéshez és a lehető legkisebb hibával történő követéshez. Ahhoz, hogy a robot képes legyen pozíciójának meghatározására, a saját és a körülötte lévő (változó) környezetből vett információkat kell alapul venni, mely alapján a pozíció értelmezésének módját két nagy csoportra oszthatjuk: lokális vagy globális [12][35].

Globális vagy abszolút pozíció esetén a robot korábbi pozíciójának értéke nem befolyásolja az új, számított pozíció értékét, hiszen az aktuális helyzet meghatározása a környezetből érkező, valamilyen külső eszköz, jeladók által biztosított információ feldolgozásával történik. Ilyen külső eszköz lehet lézeres jeladó, szatellit (GPS, GALILEO) vagy egyéb aktív jelen/jelerősségen (Bluetooth, WiFi) alapuló megoldás.

A lokális vagy más néven relatív megközelítés ezzel szemben mindig az előző pozícióhoz viszonyítva adja meg az új helyzetet, feltételezve valamilyen kezdő, kiindulási vagy úgynevezett viszonyítási koordinátát. A robot szenzorai (mozgásérzékelő, gyorsulásmérő, gyroscope stb.) mérik az elmozdulás irányát, elfordulás szögét és/vagy a megtett utat, mely alapján a relatív pozíció gyorsan és hatékonyan (olcsó műveletek árán) meghatározható.

A globális megoldás előnye tehát a lokálissal szemben, hogy nem felételez semmilyen kiindulási pontot a pontos helymeghatározáshoz, mely a gyakorlatban sokkal jobban használható, ráadásul szenzorfúzió segítségével még tovább pontosítható a lokalizációs eljárás, mely lehetőséget ad az egyes érzékelők korlátainak kiküszöbölésére is. Lokális módszer során a legnagyobb bizonytalanságot az eszközök pontatlansága és az abból vett kumulált értékek jelentik, így ezt általában csak rövid távolságoknál, globális lokalizációs eljárások között szokták alkalmazni.

Fontos azt is megemlíteni, hogy mind a globális, mind a lokális helymeghatározásnál a szenzorok által keltett bizonytalanság, az azokon megjelenő zaj, zavar vagy a környezetben megjelenő rontó tényezők (időjárást, szabad vagy fedett terület, fény, tükröződés, zavaró jelek) is jelentősen módosíthatják, torzíthatják az

eredményt, ezért a minőséget több szenzor együttes használatával (szenzorfúzió) szükséges biztosítani.

2.1.1. Odometria

Az odometria a legegyszerűbb és legalapvetőbb lokális helymeghatározási módszer, eredete a görög hodos (utazás) és metron (mérés) szavakból származik. Ahogy azt a görög szavak is sugallják, lényege az időben egymást követő – kerekek forgásából következtetett – elmozdulás mértékeknek a becslésén alapul a beépített szenzoroktól érkező információk alapján a kezdeti kiindulási koordinátát figyelembe véve. Az odometria a motorhoz kapcsolt encodertől érkező értékek gyors és egyszerű, néhány műveletes kiértékelésén alapul, mely lehetővé teszi, hogy a vezérlő hatékonyan reagáljon az eseményekre. Hátránya azonban pont egyszerűségéből fakad. Nem elég az encoder pontossága és magas mintavételezési frekvenciája, mind a belső (determinisztikus vagy rendszeres hibák), mind pedig a külső tényezők (nem-determinisztikus vagy rendszertelen hibák) is meghatározzák működése sikerességét. Belső tényezőn a motorhoz kapcsolt kerekek minőségét, nagyságát, átmérőjét, egymáshoz viszonyított eltéréseit, a fogaskerekek minőségét, a felfüggesztés precizitását vagy a hajtás fajtáját (differenciális, szinkron, Ackermann stb.) értjük. Külső tényezőnek számít a közeg, a talaj felülete és az ebből adódó súrlódás vagy tapadás hiányából fakadó pontatlanság [15].

Az odometria mint helymeghatározási technika az előbbiek alapján önállóan nem vagy csak ritka eseteben alkalmazható, hiszen az időben integrált elmozdulások során keletkező akkumulált hiba a relatív helymeghatározást hosszú trajektória esetén használhatatlanná, helytelenné teszi. Igaz ez az úgynevezett proprioceptív szenzorok teljes családjára is, amelynél a fizikai adatok (skaláris vagy vektoros értékek) a robot saját viszonyítási rendszeréből, a rászerelt érzékelőktől származnak [12].

2.1.2. Vizuális odometria

A vizuális odometria (VO) a robotika és a gépi látás együttes fejlődésével, az 1980-as évektől megjelenő és használt jelentős technológia a navigáció és lokalizáció területén. Működése az emberi látás analógiájára épül, amely során a környezet és az abban bekövetkező változások (környezet tulajdonságainak megváltozása) szolgáltatják az információt a pozíció és orientáció meghatározásához, a megtett út rögzítéséhez [8].

Lényege a robotra szerelt kamerán és annak útvonalán alapul. A kamera által szekvenciálisan szolgáltatott képeken végbemenő változás határozza meg a mozgás útját és irányát az időben. Az odometriával ellentétben itt tehát már nem a saját mozgást detektáló érzékelőktől kapott információ, hanem a külső környezet megfigyeléséből, az abból kinyert jellemzők és azok elmozdulásai (pl.: a képpont változások) biztosítják a rendszer működését. Az ebbe a családba tartozó szenzorokat, eljárásokat exteroceptív érzékelésnek hívjuk [12].

Bár a VO az inkrementált pixel változásokból még mindig csak relatív helymaghatározást tesz lehetővé, a módszer sokkal hatékonyabban használható, mint a proprioceptív érzékelésen alapuló eljárások. Ennek oka, hogy ugyan az eljárásból adódóan itt is jelentkezik a kumulált hiba – pályáról való sodródás (drifting) jelenség –, az információ gyűjtés azonban már nem függ a robot belső szerkezetétől, mechanikai és fizikai jellemzőitől, a talaj minőségétől. Így míg az odometria esetén a hibaarány akár 20% fölötti is lehet, itt néhány százalékban kimerül annak értéke. A VO további előnye, hogy elérhető bármilyen mozgási rendszerrel ellátott platformon és a tetszőleges mozgás 6 szabadságfokra is kiterjedhet. Fontos ugyanakkor hangsúlyozni, hogy ez a megoldás sem tökéletes. Homogén vagy kevésbé textúrázott környezetben jelentős a hatékonyságvesztés, hisz éppen a jellemző vonások hiányában képtelen lesz a relatív pozíció meghatározására. Túlzottan dinamikus környezetben szintén problémás lehet a használata, valamint jelentősen romlik a hatása nem megfelelő megvilágítás, árnyékok vagy elmosódás esetén is, illetve azokban a helyzetekben, ahol az egymást követő frame-ek nem megfelelő átfedéssel rendelkeznek.

2.1.3. GPS

Az érzékelők harmadik kategóriájába tartozó szenzorok az előbbiektől eltérően már globális pozíció meghatározásra is képesek. Nem a korábbi helyzetekből, elmozdulásokból következtetnek, hanem teljesen önálló, robottól független rendszerhez viszonyítva határozzák meg az aktuális pozíciót, orientációt. Ezek a megoldások adják a legpontosabb értékeket, jóllehet ár-érték arányukat tekintve az előző két kategóriába tartozó megoldásokhoz képest drágábbnak számítanak [8].

Az egyik leginkább elterjedt globális helymeghatározást lehetővé tevő rendszercsalád a műhold alapú Globális Navigációs Műholdrendszerek (Global Navigation Satellit Systems – GNSS), melynek legismertebb tagja a Globális Helymeghatározó Rendszer (Global Positioning System – GPS), melyet az Amerikai Védelmi Minisztérium megbízásából fejlesztettek ki az 1970-es évek elején. Működése a 24 Föld körüli orbitális pályán keringő műholdból áll, melyek mindegyike alacsony frekvenciás rá-

dióhullámot bocsát ki, amit a földi vevők vesznek és értelmeznek. A műholdak jelei pontos időt, pozíciójukat és egyéb adatot sugároznak. Trilateráció segítségével 3 (hiba korrigálásához 4) műhold segítségével meghatározható a beérkező jelek alapján a háromdimenziós koordináta, továbbá lehetőséget ad a sebesség és idő mérésre is [12].

A GNSS család ma már 3 működő, egymást kiegészítő és pontosító rendszerből áll: az amerikai GPS-ből, az orosz GLONASS-ból és a végül csak néhány év késéssel 2016-ban elindított európai megoldásból, a GALIELO-ból. Ez utóbbi fejlesztésének és bevezetésének alapját az adta, hogy bár a GPS-t kezdetben katonai célokra használták, egyre több polgári alkalmazása is megjelent (mezőgazdaság, geodézia, közlekedés, ipari alkalmazások stb.). A GPS-t a kezdeti felhasználást követően mesterséges pontatlansággal látták el, hogy elkerüljék és minimalizálják az ellenséges felhasználás lehetőségét (Selective Availability – SA). Ezt végül 2000 májusában az amerikai kormányzat feloldotta, továbbá megteremtette annak lehetőségét, hogy az SA képes műholdak cseréje meginduljon, így biztosítva hogy a jövőben ne is implementálhassák a megoldást. Az Európai Unió ugyanakkor továbbra is fontosnak látta a független, de a többi rendszerrel együttműködő pontosabb, biztonságosabb és így nagyobb lefedettséget, megbízhatóságot adó rendszert fejlesszen. A földi vevők ma már képesek a 3 rendszer jeleit egyidejűleg fogadni és feldolgozni, de a GNSS család a közeljövőben várhatóan egy negyedik családtaggal, a kínai fejlesztésű helymeghatározó rendszerrel fog bővülni (BeiDou).

2.2. Lokalizációs megoldások összefoglalása

A lokalizáció - legyen az lokális vagy globális -, a robotika, azon belül is az autonóm navigáció fundamentális kérdése. Hiányában vagy hibás működése esetén a feladatok végrehajtásának meghiúsulása is bekövetkezhet. Vannak részfeladatok, ahol elég a lokális pontosság, de az esetek többségében a lokális értékek állandó korrigálása szükséges a külső rendszerek által nyújtott globális koordinátákkal. Nincs tökéletes megoldás, összességében azonban elmondható, hogy a szenzorok együttes alkalmazása a költség és hatékonyság ésszerű tengelyén mozogva biztosítja a lehető legpontosabb eredményt. Az alábbi táblázat néhány szempont szerint mutatja a korábbi technikák alapján bemutatott szenzor-kategóriák közti különbséget.

Típus	Proprioceptív	Exteroceptív	External
Szenzor elhelyezkedése	Roboton	Roboton	Roboton kívül
Események forrása/hatása	Robotból / Robotban	Robotból / Külvilágban	Külvilágból / Roboton
Lokális helymeghatározás	X	X	
Globális helymeghatározás			X
Eszköz típusok	Aktív / Passzív	Aktív / Passzív	Aktív
Használat	Beltér / Kültér	Beltér / Kültér	Kültér
Példa eszközök	mozgásérzékelő , gyorsulásmérő, gyroscope	Kamera (Mono, Sztereó, RGB-D), Távolság érzékelő (RADAR, LIDAR)	GNSS
Egyéb	Kis elmozdulások (pontatlan) detektálása, gyakori mintavételezéssel; olcsó eszköz és gyors számítás	Kis elmozdulások (kevésbé hibaérzékeny) detektálása, gyakori mintavételezéssel; alacsony vagy közepes árkategóriájú eszköz, de műve-letigényesebb számítás	Drága eszköz, kis elmozdulások detektálására alkalmatlan

1. táblázat. A helymeghatározás szempontjából csoportosított érzékelők és tulajdonságaik

3. A vizuális odometria alapjai és eszközei

3.1. A vizuális odometria fogalma, építőelemei és elterjedése

Mint ahogy azt a 2.1.2. fejezetben már tárgyaltam, a vizuális odometria a kamera útvonala alapján létrehozott pixelváltozásokon alapuló eljárás, mely a lokalizáció igen pontos meghatározását teszi lehetővé [28][32]. Eszerint a fogalom szerint az eljárás alapvető építőelemeit 3 részre bonthatjuk : magára a kamerára, az általa rögzített képre, képszekvenciára és az azokon műveletet végző algoritmusokra.

A vizuális odometria elterjedésében – közvetve vagy közvetlenül – nagy szerepe volt az előbb felsorolt építőelemeknek. Maguk a képek rengeteg információt hordoznak (nehézség az értelmezésükből fakad), képek alapján tájékozódik az emberek nagy többsége is, így kézenfekvő volt, hogy a lokalizációs és az autonóm navigációs eljárások alapját képezzék. A digitális eszközök és a felépítő CCD-k rohamos fejlődése és árcsökkenése szintén predestinálta az eljárás térhódítását és növekvő szerepét. A hatékonyabb algoritmusok, és az azok gyorsabb végrehajtását lehetővé tevő processzorok, grafikus egységek (GPU) megjelenése, az adatátvitel sebességének növekedése, a memória és háttértár elérési idejének és kapacitásának bővülése szintén hozzájárult népszerűségéhez [2].

Felhasználási területe igen jelentős, nem csak az autonóm navigációban, hanem a hétköznapok egyéb tevékenységeiben is jelen van. Az orvostudomány, a közlekedésben ismert vezetés támogató rendszerek (sávelhagyást figyelő, holtérfigyelő, vészfékező rendszerek), de a kiterjesztett vagy virtuális valóság is elképzelhetetlen nélküle.

3.1.1. A kameratípusok

A VO egyik kulcseleme a látott képen kívül, az azt rögzítő kamera vagy kamerarendszer, melynek megválasztása jelentősen befolyásolja a feldolgozható információk mennyiségét, minőségét, az elvégezhető leképezést, a mozgásváltozás számítást (algoritmizálást), azaz a vizuális helymeghatározás robusztusságát és hatékonyságát. Az alkalmazott kamera típusait tekintve beszélhetünk sztereó, mono, körbelátó (omnidirectional), vagy RGB-D eszközökről [2][28][40].

Az egyetlen egy kamerával megvalósított rendszer hátránya, hogy a látott kép és a kamera pozíciójából nem lehet a pontos metrikus távolságokat meghatározni, így a trajektória csak becsülhető, skálázott értékek nem társíthatók mellé jük. Mivel a közvetlen mélységi adatok nem állnak rendelkezésre, így a mono rendszerre épülő

VO implementációk sokkal összetettebbek. Ennek ellenére széles körben használják, ami elsősorban a robotikában is érvényesülő minimalizálásnak köszönhető [40].

Sztereó kamera esetén a két kép lehetővé teszi, hogy a becsült ponthoz háromszögelés útján metrikus mélységi értéket definiáljunk. Ez sokkal gyorsabb eljárást tesz lehetővé, még akkor is, ha a képpárokat előzetesen közös síkra kell illeszteni (rektifikáció), ugyanakkor a két kamera közti távolság (alapvonal-baseline) és a felbontás befolyásolhatja a kapott értéket. Általánosságban elmondható, minél nagyobb az alapvonal (50-60 cm), annál pontosabb a társított érték. Ez természetesen azt is jelenti, hogyha túl közel vannak a sztereó kamerák, akkor elveszítjük előnyüket és mono kameraként fognak működni.

Az RGB-D szenzorok előnye mind a mono, mind a sztereó kamerával szemben, hogy közvetlenül szolgáltatnak mélységi adatokat, anélkül, hogy a VO algoritmusoktól extra számítási időt és kapacitást követelnének meg. Ezt kétféle módon képesek biztosítani: strukturált fényvetítés vagy Time of Flight (ToF) megoldás segítségével [1].

Strukturált fényvetítés esetén a kamera mellé egy infravörös (IR) fényforrást is csatolnak, mely egy előre meghatározott mintázatot (pl.: rács vagy háló) vetít az objektumra. Ez a mintázat rávetül a tárgyra, felveszi annak alakját, azaz az eredeti mintázat az alakzat geometriája szerint torzításra kerül. Az így kapott deformált mintát az egységre szerelt mélységi kamera érzékeli és kiolvassa. A módszer sikeressége nem függ a megvilágítástól, a textúrázottságtól vagy a színintenzitástól.

A Time of Flight módszer a lézeres- és ultrahangos távolságmérés speciális megoldása, mely hasonlóan az előzőhöz szintén egy extra beépített fényforráson (infravörös) alapul. Itt azonban már az általa kibocsátott fény kibocsátási és az objektumról való visszatérési idő különbségének feléből, valamint az alkalmazott fény sebességének terjedéséből kiszámítható megtett út, azaz a kamera és az objektumok közötti távolság adja a módszer lényegét. Az eljárás erős háttérvilágítás esetén nehézkessé válik, ugyanis a visszaverődő fény a környezet megvilágításával keveredni fog.

Microsoft Kinect

A Microsoft 2010-ben mutatta be a legújabb Xbox játékkonzoljához csatolható mélységi kameráját, mellyel elsődleges célja a játékkonzol iránt érdeklődő vásárlók megnyerése volt. Specifikációja, funkcionalitása és árazása alapján azonban hamar kiderült, hogy több célcsoport is érdeklődik a termék iránt (Augmented reality, virtual reality, Computer vision), továbbá a kutatásokban való alkalmazása is számottevővé vált.

Az első generációs Kinect szenzor kamerája (amely az Xbox 360-hoz készült) a mélységi adatok rögzítésére az előzőleg bemutatott strukturált fényvetítés technológiát alkalmazta. A beépített RGB kamera mellett – mely 640 x 480 felbontásra volt képes –, egy infravörös (IR) projektort és egy CMOS szenzort is elhelyeztek az eszközön. Az IR fényforrás 30 fps sebességgel működött, a mélységi adatokat pedig 320 x 240 pixel felbontásban adta vissza. Hatékony érzékelési tartománya 0.4m – 4m között volt. Ezek mellett rendelkezett négy mikrofonnal, valamint egy finompozicionálást lehetővé tevő motorral [4].

A második generációs modellnél (Kinect v2), amely 2013-ban jelent meg, a mélységi adatok meghatározását már ToF alapon oldották meg, továbbá növelték az RGB (1920 x 1080) és a mélységi kép felbontását is (514 x 424). Érzékelési tartománya jelentősen nem változott, a maximális érték fél méterrel megnövekedett. A felhasználók egészen 2017-ig várták az újabb verziót az eszközből, végül mivel a termék az elvárások alatt teljesített, a piacon befejezték támogatását és gyártását.

2018 közepén a Microsoft az Azure Cloud technológiára támaszkodva a Kinect továbbfejlesztett, extra változatát mutatta be Azure Kinect Developer Kit néven. Ezt a terméket elsősorban fejlesztői és vállalati szegmensben értékesítik, nem titkolt célja pedig a mesterséges intelligencia témaköréhez kapcsolódó fejlesztések támogatása, kiszolgálása. A mélységi kamera 1MP adatokat szolgáltat ToF alapon, az RGB képek felbontása pedig már 12MP. A hang- és beszédfelismerés támogatásához 7 mikrofonból álló alrendszert integráltak az eszközbe, az eszköz térbeli követéséhez pedig gyorsulásmérő (IMU) áll a fejlesztők rendelkezésére [5].

A Kinect (v1, v2, Kinect for Windows) és az új generációs Azure Kinect mindegyike jelentős SDK (Software Development Kit) készlettel rendelkezik. Bár az előbbiek támogatása – a gyártás befejezésével egyidőben – véget ért, kisebb közösség rendszeresen ad ki javításokat, így továbbra is elérhető Windows, Linux és MacOS alapú rendszerekre egyaránt. Az Azure Kinect támogatása viszont sokkal jelentősebb, Windows és Linux rendszerre is többféle fejlesztőkészlet és API (Application Programming Interface) áll rendelkezésre.

Asus Xtion Pro Live

A kutatások, tudományos értekezések másik nagy csoportja az Asus – Kinecthez hasonló – termékét használja. A Xtion Pro Live RGB kamerával, mélységérzékelővel és 2 db mikrofonnal ellátott egység [3]. A Kinect-el összehasonlítva két paraméterben vehetünk észre jelentősebb különbséget: az egyik a mélységi kép felbontása, valamint mintavételezési frekvenciája, a másik pedig a fogyasztása.

Az Xtion Pro Live mélységi képeit VGA és QVGA felbontásban is képes megadni, utóbbi esetben ráadásul 60 fps lesz a mintavételezés frekvenciája. RGB képe 1280 x 1024-es felbontású. A másik érdekes paramétere a fogyasztás: míg a Kinect 15 W-ot is fogyaszthat, addig a Xtion Pro Live-nak kevesebb mint 3W a fogyasztása. Ez lehetővé teszi olyan (mobil) robotokon való hosszabb távú használatát is, ahol csak akkumulátoros üzemmód elérhető (UAV-okon, drónokon stb).

Intel RealSense D400 sorozat

Az Intel RealSense mélységi kameracsalád (D400) a harmadik olyan eszközkészlet, amely az utóbbi években jelentős szerepet kapott és így a kutatási, tudományos alkalmazások meghatározó eszköze lett. Sikerét felépítésének, architektúrájának, valamint hatékony algoritmusainak köszönheti: materiálisan két egymásra épülő egységből áll, ám ezeket nem feltétlen kell egy eszközként használni, önállóan is alkalmazhatóak. Harmadik „láthatatlan" eleme pedig az a cross-platform fejlesztőkészlet, amely számos nyelven, gyors, innovatív alkalmazások elkészítését teszi lehetővé.

Az eszköz a mélységi modulból és az ahhoz kapcsolható ún. Vision processzorból áll. Mindezeket a kameraház foglalja magába, amely a belsejében alkalmazott modul és processzor változatokhoz igazodva többféle kialakítású is lehet. Beszélhetünk az Intel eszközénél is aktív, vagy passzív megoldásról: mindegyik verzió alapját a sztereó képalkotók adják, amelyet opcionálisan egészíthet ki egy infravörös projektor és/vagy RGB kamera [17].

A szenzorok által előállított nyers adatokat (legyen az akár kiegészítve egy infravörös projektorból nyert adatokkal vagy nem) a nagy teljesítményű D4 Vision processzor felé továbbítja. A D4 egy alkalmazásspecifikus integrált áramkör (ASIC – Application Specific Intergated Circuit), amely mélységi képsorozatot állít elő a bal és a jobb pixelpontok korrelálásával és korrigálásával. Ehhez nélkülöz mindenfajta host vagy egyéb GPU által biztosított processzoridőt. A D4 Vision egység felépítésére is két változat létezik, egy alaplapra integrált megoldás, a másik pedig önálló kártyaként használható.

A modulok verziói nemcsak a sztereó kamerapárokat kiegészítő eszközöktől tér el, hanem azok technikai jellemzői is különböznek, valamint integráltságuk is eltérő. Egyik szélesebb látómezővel, de kisebb felbontással és globális zárral rendelkezik (D430), míg a másik szűkebb mezőt képes detektálni nagyobb felbontással, de rolling zárral (D415). Ezek a paraméterek mind-mind más alkalmazási lehetőséget teremtenek [16].

A kameratípusok széles skálája és erőteljes fejlődése is mind azt mutatja, hogy

a tudomány az iparral karöltve közösen keresi az autonóm navigáció egyik legnagyobb kihívására – a környezet, valamint a (kül)világ leképezésére és megértésére – az eszközszintű támogatást, megoldást. Az eszközök által biztosított funkciókban is jelentős változás történt: míg korábban az adat egy/több dedikált képrögzítő segítségével került előállításra, ma már összetett módon, több kiegészítő szenzor összehangolt szinkronizációjával valósítják meg (IMU, LiDAR).

A kameraeszközök széles skáláját és alapvető paramétereit a 2. táblázat foglalja össze tömören képalkotók és típusok szerint, a felsorolt kameramárkák különböző változatait pedig a 2. ábra mutatja.

Képalkotó / szenzor Tulajdonságok	Mono	Sztereó	RGB-D	
			Strukturált fényvetítés	Time of Flight
Eszköz típusa	Passzív	Passzív	Aktív (IR)	Aktív (IR)
Mélységi adat	Nem elérhető	Közvetlenül nem, csak szoftveres számítás útján	Számított, hardveren	Közvetlenül
Mélységi pontosság	N/A	cm	micro-mm	cm/mm
Árkategória	+	++	+++	++ (+++)
Eszköz mérete	Kicsi	Közepes	Nagy	Nagy
Példa			Kinect v1	Kinect v2 / Intel Real Sense család

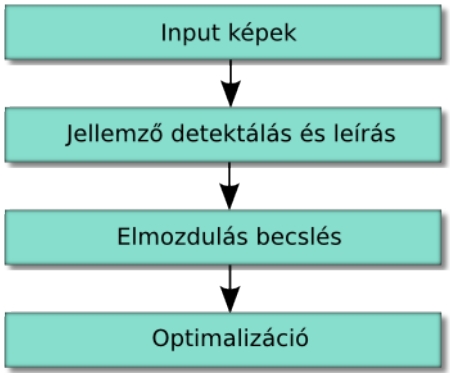
2. táblázat. Mono, sztereo és RGB-D kamerák jellemzői

2. ábra. A bemutatott RGB-D kamerák [4] [3] [17]

3.1.2. A vizuális odometria algoritmusa

A vizuális odometria algoritmusának legfontosabb feladata, hogy a kamera mozgásának alapján leírja az egymást követő relatív mozgásváltozásokat, transzformációkat. Segítségével egzakt kiindulási koordináta nélkül meghatározható a teljes trajektória (kumulált relatív mozgások által). A művelet végezhető valós időben, azaz a képek rögzítésével egyidejűleg vagy offline módon valamennyi kép rögzítését követően. Mindezek alapján az algoritmus az alábbi lépésekben definiálható (iteratív módon) amit kivonatosan a 3. ábra is szemléltet[28][32][33]:

1. a mozgást leíró, egymást követő képek rögzítése
2. a képek előfeldolgozása (sztereó kameraképpároknál erre mindenképpen szüksége van)
3. a képek tulajdonságainak detektálása
4. a detektált tulajdonságok leírása, reprezentálása
5. az egymást követő képeken meghatározott tulajdonságok megfeleltetése, párosítása vagy követése
6. mozgás becsélese, transzformációs mátrix meghatározása (a transzformációs mátrix minden esetben egy rotáció és transzlációs komponensből áll)
7. validáció és optimalizáció



3. ábra. A VO folyamatának fő lépései

A mozgást leíró képek rögzítése és előfeldolgozása

A mozgást leíró képek rögzítése egymást követő diszkrét időpillanatokban történik. Mono kamera esetén minden pillanatban egy, sztereó esetén a jobb és bal kameráknak köszönhetően szinkron módon kettő.

A képek előfeldolgozása minden esetben nélkülözhetetlen, korrigálni kell a lencse torzításait (pl.: radiális torzítások), előszűrést kell végezni rajtuk, valamint a sztereó képpárok esetén, ha a kamerák nem egy síkba esnek, el kell végezni az epipoláris geometrián alapuló standardizációt (rektifikációt) azért, hogy teljesen igazított képeket kapjunk és így a mélységre vonatkozó értékek egyszerűbb módon meghatározhatóak legyenek. Rektifikáció során – mivel a kamerapár kalibrációja és az egymástól vett relatív helyzete előre definiált, vagy kiszámítható – a képek egy tulajdonképpeni közös, a kamera alapvonalával párhuzamos képzeletbeli síkra történő vetítését végezzük el. Így érjük el, hogy olyan igazított képeket kapjunk, amelyeken a bal kép egy adott pontja pontosan ugyanabba a sorba esik mind a két képen. Ezáltal a tulajdonságok meghatározása már sokkal gyorsabban és egyszerűbben megtörténhet, mellyel az algoritmus további lépései, kiváltképpen is a tulajdonságok megfeleltetése gyorsítható jelentős mértékben.

A képek tulajdonságainak detektálása, leírása, reprezentálása és megfeleltetése

A képek tulajdonságainak meghatározása, leírása nem egyszerű feladat. Egy-egy pont, feltűnő átmenet könnyedén megkülönböztethető környezetétől, de leírása, jellemzése és a több képen történő összehasonlíthatósága, beazonosítása sok tényezőtől függ. A képek zajtól terheltek, a mozgás közben változhat az orientáció, arány és akár a megvilágítás is eltérő lehet. Ezek a problémák, valamint a rendelkezésre

álló számítási kapacitás és a feladat összetettsége mind-mind befolyásolják a művelet pontosságát, hatékonyságát, alkalmazhatóságát, amely így hatással van a teljes feldolgozási folyamat sikerére így megválasztásuk kiemelten fontos feladat .

A tulajdonságok és leíróik képeken át történő meghatározására két megközelítést különíthetünk el: a direkt (közvetlen) és az indirekt (közvetett) alapú eljárásokat [28][32].

Direkt vagy megjelenés alapú (appearance-based) megoldásról akkor beszélhetünk, ha a mozgás követéséhez közvetlenül a pixelintenzitások értékeit használjuk fel. Ez lehet valamennyi pixel alapján (sűrű – dense), történhet egy bizonyos küszöb fölötti értékkel rendelkező pixelek szerint (semi dense) vagy akár csak néhány kiemelt alapján (ritka sparse). Az eljárás alkalmazható a kép egészére vagy egyes részleteire vonatkozóan is célja, hogy a transzformációs mátrix meghatározásához minimalizálja az intenzitáskülönbség hibákat. Ekkor a (rész)terület valamennyi pixel intenzitásértékét figyelembe vesszük, így az egymást követő képek transzformációs mátrixának alapjait lényegében intenzitásgradiensek (nagyság és irány) eltérései fogják biztosítani. A direkt megoldás egyik leginkább ismert módszere a minta keresés (template matching vagy correlation). Lényege, hogy a soron következő képen kiválasztásra kerül az aktuálisból választott jellemző részre leginkább hasonlító terület. A hasonlóság számítása SSD (Sum of Squared négyeztes különbség) vagy NCC (Normalized Cross Correlation) segítségével történhet, a megfeleltetett rész pedig a legnagyobb értékkel rendelkező lesz.

Ezzel szemben a közvetett (indirekt) vagy tulajdonság alapú (feature-based) technika esetén nem a teljes képet, hanem annak csak lokális részleteit tekintjük. Pontosabban definiálva olyan szembetűnő tulajdonsággal rendelkező lokális pontokat, vonalakat, alakzatokat keresünk a képen (sarok, élek, foltok), melyek feltehetően a további képkockák során is meg fognak jelenni és így rajtuk keresztül könnyen kalkulálható a transzformációs mátrix. A lokális jellemző rendszerint valamilyen képtulajdonság, vagy több képtulajdonság egyidejű megváltozásával azonosítható. Mint ahogy azt az előbbiekből is mutatják, a jellemző pont meghatározása nem feltétlen korlátozódik egy dedikált pixelpontra, hanem annak milyenségétől függően a jellemző pont leírhatja annak közvetlen környezetét is [33].

A detektorok legfontosabb általános jellemzői a pontosság, gyorsaság és hatékonyság, valamint a robusztusság. Kiemelten fontos az ismételhetőség is, azaz ha a detektor ugyanarról a jelenetről két képet kap (de más a megvilágítás vagy a nézőpont) akkor is a lehető legtöbb közös pixelt (keypoint) és azok közvetlen környezetét jelölje ki (descriptor), ahol kirívó a képtulajdonság változás. Mindehhez biztosítani kell, hogy a detektor bizonyos képműveletekre (affin transzformációk, átméretezés,

forgatás) invariáns legyen.

A tulajdonságok leírása a későbbi összehasonlítás céljából a descriptorok segítségével történik. A descriptor a pixel és környezetének információját dolgozza fel és rendszerint vektoros formában tárolja, reprezentálja. Annál jobb a leírás, minél kisebb a kinyert információ mérete (vektor hossza), de maradéktalanul jellemzi az adott pontot, területet.

A feature-based alapú detektorok és leírók kategorizálása többféle módon is történhet, leginkább azonban a detektálandó jellemzőpont típusa szerint vagy az alkalmazott eljárás alapján csoportosíthatjuk őket. Előbbiek szerint beszélhetünk valamilyen kontúr/görbület (sarok, él, folt) alapú megoldásról, utóbbi esetben megkülönböztetünk bináris leírókat, intenzitás/gradiens alapú detektorokat és leírókat, valamint minta (patch) alapú megoldásokat. A következő részben a legismertebb detektorok közül a vizuális odometria során leginkább alkalmazott eljárásokat szeretném röviden kiemelni és bemutatni.

Harris sarokdetekor

A sarokdetektáló algoritmusok legelső és legismertebb megoldása Harris és Stephens nevéhez köthető, akik Moravec algoritmusát pontosították és publikálták 1988-ban. Az algoritmus lényege abban áll, hogy egy lokális ablakon belül valamilyen súlyfüggvény segítségével minden egyes pixelre vonatkozóan megvizsgálja az intenzitásváltozásokat mind a 8 irányba. Eszerint az intenzitáskülönbség 3 esetét lehet így megkülönböztetni [18][25]:

- nincs intenzitásváltozás semelyik irányba (homogén terület)
- jelentős változás minden irányba (sarok vagy izolált pont)
- él esetén jelentős változás csak az ablak élre merőleges irányú elmozdulása esetén tapasztalható

Az intenzitáskülönbség matematikai formalizmusát az alábbiak szerint definiálták:

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u, y+v) - I(x,y)]^2 \tag{1}$$

ahol az

- E a hibafüggvény az eredeti és az eltolt ablak között

- $w(x, y)$ az ablakfüggvény x, y pontban
- $I(x, y)$ intenzitás x, y pontban
- $I(x + u, y + v)$ az $[u, v]$ eltolás szerinti intenzitásérték $(x + u, y + v)$ pontban

Az alkalmazott w függvény lehet négyszögletes (1 vagy 0) vagy Gauss-típusú súlyfüggvény is. Ahhoz, tehát hogy változásokat – éleket és sarkokat – detektáljunk azokat az ablakértékeket kell keressük, ahol jelentős intenzitásváltozás figyelhető meg kis u, v eltolás esetén, azaz tehát maximalizálni kell a $E(u, v)$ függvényt.

Az $E(u, v)$ számítása időigényes minden egyes pixelre vonatkozóan (függ az ablakmérettől és a képpontok számától) ezért közelíteni szokás Taylor sorfejtést alkalmazva, amellyel az alábbi összefüggésre jutunk elsőrendű közelítést figyelembe véve (jelölje a parciális deriváltakat I_x és I_y):

$$E(u, v) \approx \sum_{x,y} w(x, y)(u^2 I_x^2(x, y) + 2uvI_x(x, y)I_y(x, y) + v^2 I_y^2(x, y)) \tag{2}$$

amelyet mátrix alakba írva, az alábbi kifejezést kapjuk:

$$E(u, v) \approx \begin{bmatrix} u & v \end{bmatrix} \left(\sum_{x,y} w(x, y) \begin{bmatrix} I_x^2(x, y) & I_x(x, y)I_y(x, y) \\ I_x(x, y)I_y(x, y) & I_y^2(x, y) \end{bmatrix} \right) \begin{bmatrix} u \\ v \end{bmatrix} \tag{3}$$

Így tehát ahelyett, hogy minden képpontban a négyzetes különbség (SSD) értékeket határoznánk meg, egy lokális ablakkörnyezetben súlyozva összegezzük a parciális deriváltakat, így az eredmény minden egyes pontban egy 2 x 2-es mátrix lesz. Ezt a továbbiakban jelölje M.

Az M mátrix sajátértékei jól jellemzik az adott pontot, ezt alapul véve Harris és Stephens egy sarkossági mérőszámot határozott meg, mely leírja egy ablak jellemzőjét a lehetésges sarokpontra vonatkozóan. A sarkossági mérőszámot jelölje R, értékét az alábbi képlet adja meg:

$$R = det(M) - k(trace(M))^2 \tag{4}$$

ahol k egy empirikus módon választott konstans 0.04 és 0.06 között.

$$trace(M) = \lambda_1 + \lambda_2 \tag{5}$$

$$det(M) = \lambda_1 \lambda_2 \tag{6}$$

A sajátértékek számítása helyett 2 x 2-es mátrix esetén az M nyomát számolhatjuk a főátlóban lévő elemek összegével, míg determinánsát a fő átlóban és a mellékátlóban lévő elemek szorzatának összegével. Eszerint tehát a sarkosság értékét az M mátrix sajátértékei befolyásolják aszerint, hogy:

- ha mind a két sajátérték kicsi, akkor $|R|$ is kicsi - homogén területet jelöl
- ha egyik sajátérték nagyobb mint a másik, akkor $R < 0$ - élről beszélhetünk
- ha a két sajátérték nagy és közel azonos, akkor R nagy, ami sarkot jelöl.

Sobel operátor - Éldetektálás

A Sobel – közelítő elsőrendű derivált – operátor kétdimenziós szürkeárnyalatos képen végez gradiens nagyság számítást egy 3 x 3-as konvolúciós maszkpár segítségével (3). A maszkpár x és y irányba (oszlop és sor) eltérő értékeket használ, végigcsúsztatva a képen képpontról képpontra meghatározza az x és y irányú intenzitásváltozásokat, amely alapján becsülhető a gradiens magnitúdója és iránya. A kapott nagyságértékek pl.: a kép hisztogramja alapján meghatározott küszöbértéknek megfelelő küszöbölés (thresholding) után – hogy csak az igazán megfelelőket szelektáljuk – a lehetséges éleket mutatják. A magnitúdó számítása a 31. oldal (7) vagy a 31. oldal (8) képlete alapján történhet attól függően, hogy pontos vagy közelítő értékkel szeretnénk dolgozni. Az orientáció pedig a 31. oldal (9) képlete alapján számolható.

-1	0	+1
-2	0	+2
-1	0	+1

+1	+2	+1
0	0	0
-1	-2	+1

3. táblázat. Sobel maszkpár x és y irányú G_x, G_y deriváltak közelítéséhez

A 3 x 3-as maszkméret egyrészt hatékonyabb a zajokkal szemben, ugyanakkor meredek élek esetén utólagos élvékonyítás szükséges, mert azok szélesebben jelennek meg a kelleténél. Az eljárást a megválasztott küszöbérték is jelentősen befolyásolhatja, így azt érdemes iterált módon, többféle értékkel elvégezni.

$$|G| = \sqrt{G_x^2 + G_y^2} \tag{7}$$

$$|G| = |G_x| + |G_y| \tag{8}$$

$$\theta = \arctan\left(\frac{G_y}{G_x}\right) \tag{9}$$

Canny éldetektor

A Canny éldetektor (1986) John F. Canny nevéhez köthető, aki folytonos, lineáris szűrőjével olyan algoritmust alkotott, amellyel a képeken széles körben megjelenő élek (mélységi változásnál, felületek találkozásánál, árnyékok megjelenésénél) optimális, azaz minimális hibával történő felismerését teszi lehetővé. Optimális voltát 3 kritériumban definiálta:

- létező élt detektáljon, ne legyenek zaj által keltett hamis élek
- kerülje a többszörös detekciót ugyanarra az élre vonatkozóan.
- a megtalált élek helyét pedig pontosan határozza meg (ott ahol van, olyan szélesen ahogy van)

Az algoritmus végrehajtása négy fő lépésre bontható. Azért, hogy a zajszerű intenzitáskülönbségek ne fals élként jelenjenek meg az eredményben első lépésben Gauss-szűrővel simítják a képet. A Gauss-szűrő egy konvolúciós eljárás, ahol az alkalmazott maszksúlyok a vizsgált képpont környezetétől távolodva csökkennek, de összértékük mindig egy lesz. Az alkalmazott maszkméret lehet 3 x 3-as 5 x 5-ös vagy akár nagyobb is, ahol a maszkméret a Gauss-függvény közelítésének pontosságát, a szórása pedig a simítás mértékét befolyásolja (méret és szórás összefügg tehát). Minél nagyobb a szórás, annál homályosabb lesz tehát a kép, mely egyrészt befolyásolja a szűrő végrehajtási idejét, valamint hatással lesz az algoritmus precíz él-helymeghatározó képességére. Kisebb szűrő tehát kevésbé elmosódott képet ad, amelyen az algoritmus optimálisabban találja meg a finomabb (vékony, éles) éleket.

Második lépésben az intenzitás gradiensek meghatározása történik a deriváltakkal x és y irányba. Ebben a lépésben a vízszintes, függőleges és diagonális elhelyezkedésű élintenzitás változásokat is meghatározza, definiálja az élerősséget és az irányt (élorientáció), melyhez többnyire Sobel operátort használ (4 maszk), de az algoritmus továbbfejlesztése Prewitt vagy Roberts operátort is ajánl.

Harmadik lépésben a nem-maximális élek/pontok elnyomása (non-maximum supression) történik. Az eljárás tulajdonképpen az élek vékonyítására szolgál abból kiindulva, hogy a legnagyobb intenzitásváltozás (lokális maximum) az él-normális irányában van. Ehhez minden pixel élorientációját közelíteni kell 0°, 45°, 90°, 135° irányok valamelyikéhez és az kapott irány szerinti közvetlen szomszédokkal kell az vizsgált pixel élerősségét összehasonlítani. Amennyiben legalább egy szomszédja nagyobb intenzitású, úgy a vizsgált képpontot el kell hagyni, egyéb esetben viszont meg kell tartani. Ezzel tehát minden a gradiens irányába eső fals pontot eltávolítunk, és így vékonyított éleket kapunk. A gradiens irányának közelítése helyett gyakran interpolált értékekkel dolgozunk. Ha az él iránya nem esik rá semelyik környező pontra, akkor azok interpolált értékeiből határozzuk meg a gradiens nagyságát és azzal végezzük el az összehasonlítást. Ez az eljárás lassabb, de pontosabb eredményt ad a nem-maximumok kiszűrésében.

Negyedik lépésben az elnyomott pixelek nélkül leírt élek küszöbölése következik. Ezt az algoritmus a szokásostól eltérően nem egy, hanem kettő küszöbbel oldja meg azért, hogy a lehető legpontosabban definiálja az éleket és azok folytonosságát. Ezt az eljárást hiszterézises küszöbölésnek nevezzük. Tekintsünk két küszöb értéket T_{low} és T_{high} ahol $T_{low} < T_{high}$, valamint a vizsgált képpont gradiens nagyságot, ezt jelölje E_c. Amennyiben a képpont gradiense nagyobb mint felső küszöb $E_c > T_{high}$, akkor a pontot az élt leíró pixelek közé emeljük, amennyiben alacsonyabb, a pixelt elvetjük. Ha a gradiens értéke a két küszöb érték közé esik $T_{low} <= E_c <= T_{high}$ akkor csak abban az esetben rögzítjük az élhez, ha már valamely szomszédja korábban megfelelt a hiszterézises küszöbölés feltételének. A küszöbértékek megválasztása itt is fontos, jelentősen befolyásolja az alkalmazási terület, a feldolgozandó kép tartalma, struktúrája, intenzitása, színtelítettsége.

SIFT detektor és decriptor

A SIFT (**S**cale **I**nvariant **F**eature **T**ransform) algoritmus közel húsz éves, mégis a mai napig az egyik leginkább alkalmazott jellemező detektor és leíró technika. Lowe publikálta 2004-ben, népszerűségét pedig az adja, hogy invariáns a legtöbb képműveletre (forgatás, eltolás, affin transzformáció, orientáció és skálázás), a megvilágítás megváltozására pedig korlátozottan invariáns, így széles körben alkalmazható [26] [22].

A SIFT algoritmus egymást követő, magas bonyolultságú műveletek sorozatát definiálja, úgy hogy a jellemző keresés során a legbonyolultabb műveleteket csak legvalószínűbb és legstabilabb jelölteken hajtja végre, ezzel is időt és számítási költséget

takarítva meg.

Az algoritmus 4 lépést definiál. Első lépésben olyan extrém pontok keresése a cél amely kicsinyítésre és nagyításra is szélsőértékkel rendelkezik. Ehhez többszintű skálateret hoz létre, amelyhez a Differnce of Gaussian (DoG) technikát használja. Ennek során különböző léptékű Gauss simítást végez a képen, majd mindezt megismételi az input (felére) csökkentett, kicsinyített változatán is. A lépést ciklikusan tovább folytatva állítja elő a skálateret, majd a keletkezett szomszédos konvolúciós szintek különbségképeit képezi. Az extrém pontok keresése ezeken a különbségtér képeken keresztül történik, megvizsgálva minden egyes pont szomszédait mind az azonos szinten mind pedig a szomszédos szinteken. Amennyiben a képpont értéke az összes összehasonlított pixel közül a legnagyobb (lokális maximum) vagy a legkisebb (lokálist minimum), akkor azt kulcspontnak jelöli.

A választott kulcspontok halmaza igen nagy számú lehet, a feldolgozást sok esetben lassíthatják vagy tévesen befolyásolhatják, így a második lépés ezen pontok további szűrésére fókuszál. Cél, hogy ne csak kulcspont, hanem stabil kulcspont legyen a kiválasztott pixel, azaz csak azokat a pontokat vegye figyelembe, amelyek a legerőteljesebb tulajdonsággal bírnak. A választott kulcspontok eredeti képen való lokalizálása interpoláció segítségével történik. Azokat a kulcspontokat, amelyek nem szolgáltatnak megfelelő információt, figyelmen kívül kell hagyni küszöbölést alkalmazva. A kulcspont kontrasztja a környezetében található pontok intenzitásértékeinek deriváltjaiból kalkulálható, így egy jól meghatározott küszöb alapján minden zajra érzékeny, alacsony kontrasztú pont és gyenge élpont eltávolítható.

Harmadik lépésben a pontokhoz tartozó orientáció meghatározása történik, melyet a lokális gradiensek figyelembevételével valósított meg Lowe. Ez a kulcsa annak hogy forgatással szemben invariáns jellemző detektort kapjunk. A gradiens nagyságának és irányának számítása a Gauss-féle simított kép adott kulcspontja körüli szomszédos régió minden egyes pixelére megtörténik. Az így kapott értékeket súlyozzuk (Gauss maszk szerint) és 36 elemű orientációs hisztogramot képezünk. Előfordulhat, hogy a lokális gradiensek alapján egynél több domináns irány is rögzíthető, ezeket súlyozva kell figyelembe venni a fő orientáció meghatározásakor.

Utolsó lépésben a kulcspontok leírását adjuk meg, amelyet úgy alkothatunk meg, hogy megvizsgáljuk a kulcspontot és környezetét. Minden pont és környezete egy mintázatot takar, amelyet egy 16x16-os ablak további 4x4-es felbontásával vizsgálunk. Ez összesen 16 darab irányhisztogramot jelöl, egyenként 8 értékkel. Ezek összessége definiálja a leíró vektort, mely összesen egy 128 elemű számsorozat lesz.

SURF detektor és leíró

Herbert Bay 2006-ban publikálta a SURF (Speed Up Robust Feature) algoritmust, mely a SIFT-hez hasonlóan kulcspontokat keres és azok környezetét vizsgálja. A SIFT-től eltérően azonban itt a simítás nem képpiramis segítségével, hanem a szűrőméret növelésével történik, amely jelentősen felgyorsítja az eljárást. Szintén az eljárást gyorsítja, hogy az orientációk meghatározásához nem lebegőpontos, hanem diszkrét egész értékű becslést alkalmaz (Haar-wavelet) [19].

Összefoglalás

A sarokdetektáló algoritmusok közül leginkább Harris elsőként publikált detektora a legismertebb, hátránya azonban hogy nagyon érzékeny a képátméretezésre. A FAST (Features from Accelerated Segment Test) gyorsasága és egyszerűsége mellett 16 pixel csúszóablakos eljárásának köszönhetően ugyanazt az eredményt adja átméretezett képen is. Élek detektálására a Canny-féle éldetektort alkalmazzák leginkább, míg foltok meghatározása SIFT (Scale-invariant feature transform) és SURF (Speeded Up Robust Features) algoritmusok segítségével történik [9][13].

A megtalált tulajdonságok egymást követő képeken való megfeleltetésére kétfajta technika létezik. Beszélhetünk tulajdonság követésről, mely esetén az t-edik időpillanatban feldolgozott kép megtalált jellemzőit korrelációs eljárással követjük a t+1, t+2... időpillanatokban feldogozandó képeken is. Ez a kameraképek közötti kis mozgások esetén hatékony. Nagy változások közötti megfeleltetésre hatékonyabb, ha az egymást követő képek jellemzőit egymástól függetlenül kinyerjük, majd a tulajdonságokból pontpárokat képzünk a t és a t+1-dik időpillanatokban képzett képek szerint. A társítás lehet brute force alapú (mindent mindennel) vagy valamilyen hasonlósági metrika szerinti.

Az utóbbi évek kutatásai a direkt és tulajdonságalapú megoldások mellett egy harmadik, ún. hibrid megoldást is alkalmaznak. Ebben egyszerre van jelen a direkt és a feature-based alapú eljárás, együttes használatuk célja pedig az eljárások gyengeségeinek kiaknázása[28].

A direkt megoldás a teljes képet vizsgálja, így számítási kapacitása jelentősebb, ugyanakkor implementációja kevésbé bonyolult, mint az indirekt eljárást alkalmazó módszereké. Indirekt eljárások esetén ugyanis a hamis tulajdonságokat, párokat (kívülálló – outliers) szűrni kell, ami jelentősen bonyolíthatja azt. Míg a heterogén képeken az indirekt, addig az alacsony textúrázottságú környezetben (vagy az ori-

entációban kevéssé változó képeken) készülteken a direkt megoldás a hatékonyabb. Mind a direkt, mind pedig a tulajdonság-alapú módszer alkalmazható mono, sztereó vagy RGB-D kamerák esetén is.

Mozgás becslése

A mozgás becslése során az egyes időpillanatokban készített képek és az előző lépésben megtalált, megfeleltetett tulajdonságok közötti transzformációkat határozzuk meg. Ezekből az elemi transzformációkból konkatenációval kapjuk a teljes becsült útvonalat. A 1.ábra szemlélteti a VO mozgás becslésének folyamatát. A kamera pozíciók (C_k) és a detektált jellemzők alapján megadható minden egyes k időpontok közötti átmenetet leíró $T_{k,k-1}$ transzformációs mátrix [32].

A mozgás becslésének megoldásai függnek az alkalmazott kamerarendszertől és az elérni kívánt céltól (hatékonyság, költség, performancia stb.). Eszerint beszélhetünk 2D-2D, 3D-3D és 3D-2D eljárásokról.

Validáció és optimalizáció

A validációs és optimalizációs lépés a VO folyamat egyik legfontosabb része, hiszen hiába találunk rengeteg jellemző pontot/területet az egyes frame-ek között, ha azok egymásnak való megfeleltetése hibás vagy pontatlan, akkor a becsült és transzformációval leírt mozgásváltozás is tévesen kerülhet megállapításra. Ennek megakadályozására már a jellemző pontok detektálásánál és megfeleltetésénél is gondolni kell.

A megfelelő pontok azonosítására többféle megközelítés is létezik. A leginkább alkalmazott módszerek közé azok az eljárások tartoznak, amelyek a hibás értékpárok kiszűrésével (ezek az ún. outlierek) adnak megfelelő becslést a mozgásváltozásra. A másik irányzat nem a kiugró értékekre fókuszál, hanem a legnagyobb halmazát keresi a hasonló jellemzőpároknak. Előbbi megoldások sokkal inkább elterjedtek, közülük is a RANSAC (Random Sampling Consensus) vagy az ICP (Iterative Closes Point) algoritmusokat használják leginkább. Mindegyik megoldás lényege az iteratív módon végrehajtott feldolgozáson áll, az eredmény pontossága pedig függ a kiválasztott jellemzők darabszámától, a végrehajtott iterációk számától és a validációs kritérium (amely a párokat alkalmassá vagy alkalmatlanná teszi) értékétől. A hasonló jellemzőpárokon alapuló megoldások leginkább olyan megoldásoknál használhatók, ahol a

mozgás sebessége kicsi és a környezetben sincs túl sok dinamikus változás.

ICP algoritmus

Az **I**terative **C**losest **P**oints algoritmus az egyik leggyakrabban alkalmazott eljárás, mely a pontfelhők illesztését fokozatosan, iterációk sorozataként hajtja végre. Az illesztéshez szükséges transzformációt minden egyes iterációban az egymásnak megfelelő (hasonló) pontok, ponthalmazok párok közti euklideszi távolságok minimalizálásával határozza meg [24] [14].

Az algoritmus lényegében 5 lépésből áll:

- első lépésben a referencia és illesztendő pontok kiválasztása történik (selection). A pontok kiválasztása történhet véletlenszerűen vagy valamilyen sajátosság alapján (szín, görbület stb.)
- második lépésben a a korábbi pontokból kialakítandó pontpárok meghatározása történik (matching). Ez történhet brute-force alapon mindent mindennel vagy a legközelebb esők szerint valamilyen küszöbérték alapján
- harmadik lépésben a pontpárok közötti transzformáció becslés történik (estimation), a lehető legjobb illeszkedés szempontjából pl. SSD-vel. A legjobb eredmény eléréséhez ekkor szükség lehet a kiugró értékek eltávolítására is
- a legkisebb hibát adó transzformáció végrehajtása az illesztendő pontfelhőn (alignment)
- megállási feltétel alkalmazása és kilépés (exit) vagy az algoritmus ismétlése (iterate) a második ponttól kezdve új párok kalkulálásával. Megállási feltétel lehet valamilyen előre definiált iterációszám elérése vagy ha a kapott transzformációk közötti különbség kisebb egy előre definiált küszöbértéknél

Az algoritmus lépéseiből is kitűnik, hogy hatékonysága igen jelentősen növelhető a megfelelő pontok kiválasztásával és/vagy az alkalmazott összepárosítási stratégia segítségével. Az ICP mérőszáma, eredménye azt mutatja meg, hogy az illesztés mennyire volt sikeres aszerint, hogy az adott pontfelhőt, frame-t a meglévő térképünkhöz fűzzük vagy eldobjuk.

RANSAC algoritmus

A **RAN**dom **SA**mple **C**onsensus algoritmus a Monte-Carlo típusú becslési módszerek csoportjába tartozó eljárás. Ezen eljárások közös vonása, hogy az eredményt,

modellt véletlenszerű kiválasztás alapján próbálják meghatározni fokozatos finomítással, azaz egymást követő ciklusok végrehajtásával. A cél az, hogy ha az egyes iterációkban nem is érjük el a legjobb célt, a végeredmény a legjobb modell megtalálásával záródjon. Mindezekből következik, hogy RANSAC esetén az eredmény nem-determinisztikus, függ a kezdeti modelltől, az adathalmaztól, azok tulajdonságától (zaj, hibás értékek), a választott hibahatártól és az iterációk számától is [35] [14].

A RANSAC első lépésben az adathalmazból kiválasztott véletlen pontok alapján meghatározza az azt leíró modell paramétereit. Mindezt a lehető legkevesebb ponttal teszi, pl.: kétdimenziós input esetén egy egyenes illesztésekor kettő, míg háromdimenziós input esetén egy sík paramétereinek meghatározásához három pontot választ.

Második lépésben az algoritmus az adathalmaz valamennyi pontját megvizsgálja és az első lépésben meghatározott modellel való megfelelést ellenőrzi: egyenes esetén azt vizsgálja, hogy adott pont illeszkedik-e az egyenesre, ha nem milyen távol van tőle; sík esetén pedig, hogy egybeesik-e az adott pont a síkkal. Egy előre meghatározott küszöbérték alapján az így megvizsgált pontok hozzáilleszthetők a modellhez (inlier) vagy elutasíthatók (outlier). Az eredmény annál jobb, minél több pont hozzáilleszthető a modellhez, azaz a megtalált pontok konszenzusba vannak a modellel (konszenzus érték).

A cél az, hogy a véletlenszerűen választott modellek közül a legjobbat határozzuk meg, ebben segít a konszenzus érték. Minél magasabb annál jobbnak mondható a modell. Az így kiválasztott modell pontjai segítségével, azaz az inlinerek felhasználásával utolsó lépésben a modell újraszámítható és pontosítható.

Az eljárást gyakran arra is használják, hogy a nem kívánatos pontokat meghatározzák az adathalmazon, ezért a RANSAC-ot gyakran outlier-detektornak is nevezik.

Jól látható, hogy az algoritmus jósága nagyban függ az adathalmaztól és a választott küszöb értéktől. Amennyiben alacsony értéket választunk sok outlier is bekerülhet a modellbe, ami torz eredményhez vezethet. Magas küszöbérték esetén pedig előfordulhat, hogy túlságosan homogén konszenzus értékeket kapunk, ami megnehezítheti a legjobb modell kiválasztását.

4. SLAM

A szimultán lokalizáció és térképezés eljárás (Simultaneous Localization and Mapping – SLAM) a robotika és más tudományterületek egyik jelentős kutatási területe. Ahhoz, hogy a robot elvégezze a feladatát és valóban autonómmá tudjon válni, tudnia kell hol áll, merre mehet, hol járt már, milyen objektumok veszik körül és hogyan néz ki a környezete. Ez utóbbira azért is szüksége lesz, mert relatív módon ez alapján tudja megadni a választ az előtte lévő kérdésekre. Ha nem rendelkezik előre tárolt reprezentációval, akkor a szenzoros adatai alapján egyszerre kell feltérképeznie a körülötte lévő világot és az így reprezentált világ alapján lokalizálnia (pozíció és orientáció szerint egyaránt) magát. A SLAM erre ad megoldást, lehetővé téve, hogy a robot anélkül, hogy bármilyen előzetes információval rendelkezne helyzetéről, pozíciójának meghatározásával egyidejűleg képes legyen térképet építeni környezetéről [10].

A SLAM egyik legnagyobb kihívása a bizonytalanság, amely több dolgot is magába foglal [35]. Egyrészt, hogy teljesen ismeretlen környezetben kell tájékozódnia a robotnak, másrészt, hogy a környezetben detektálható tulajdonságok igen változó számúak lehetnek (hol több, hol kevesebb figyelhető meg, befolyásolja a helyszín, kontrasztosság, megvilágítottság stb.), harmadrészt, hogy a kapott szenzoros adatok sem mindig pontosak, zajtól terheltek, interferáltak, amit a mintavételezés gyakorisága is befolyásolhat, negyedrészt pedig, hogy e bizonytalan térkép alapján kell a robotnak lokalizálnia magát. Ez a bizonytalanság tehát visszahat a reprezentáció minőségére, a lokalizáció pontosságára, így a SLAM egyik kiemelt részfeladata a bizonytalanságból eredő hiba csökkentése. Ez több szinten is megvalósulhat: történhet akár a szenzorok fúziójával, vagy a folyamat részfeladatainak hatékonyságának növelésével [29]. Mindezeknek azonban ára van: hatással van az egész eljárás pontosságára, a működés optimalizáltságára, a gyorsaságra, az igényelt számítási kapacitásra vagy a felhasznált memóriára.

Jól érzékelhető tehát, hogy a SLAM számos kihívással küzd, melyek egy része az utóbbi két-három évtizedben a kutatások fókuszában állt, más részük viszont csak elméleti alapon megoldott. Kimondható ugyanakkor, hogy az egyik leggazdagabb módon fejlődő területe a robotikának, mely egyszerre próbál hidat képezni bizonytalanságból eredő problémák és a valós idejű kihívások között beltéren és kültéren egyaránt.

4.1. SLAM eljárás alapjai

A SLAM eljárás végrehajtása során három fő lépést különíthetünk el: iránypontok keresése és azonosítása, adattársítás a korábbi és az aktuális iránypontok között, valamint az állapotfrissítés [29]. Ez utóbbi magába foglalja az robot helyzetének leírását, az iránypontok pontosítását és a térkép frissítését is. A 3 lépés megvalósítása nagyban függ a robot típusától, kinematikájától, a roboton elérhető mérőeszközöktől, a környezet milyenségétől (homogenitás, textúrázottság), az iránypontok minőségétől és számától, az elérhető számítási és memória kapacitástól, a megoldandó feladattól és a felhasználás módjától (valós idejű, kritikus rendszer). A 4. ábra a SLAM alapvető folyamatát mutatja.

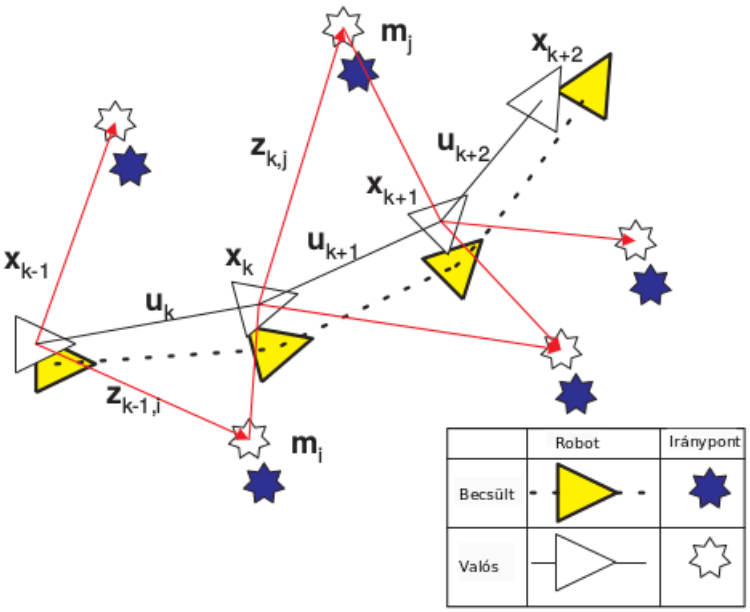
Az első és nélkülözhetetlen lépés az iránypontok meghatározása és felderítése. Iránypontoknak azokat a tulajdonságokat tekintjük, amelyek egyedileg azonosíthatók, könnyedén megfigyelhetők és újra detektálhatók, továbbá célszerű statikus pontokat választani, mert ezekhez sokkal könnyebb viszonyítani. Ilyen lehet pl.: egy ultrahangos jeladó, Az azonosított iránypontokat a szenzorok segítségével ki kell tudni nyerni a környezetből. Legyen szó proprioceptív vagy exteroceptív érzékelőről, minden esetben számíthatunk valamilyen zajra, amely hatással lesz a leképezett iránypontra és így az egész folyamatra. Minél több iránypontunk van, annál pontosabban és gyorsabban lehet azokat rögzíteni és egymáshoz viszonyítani. Ezzel pedig eljutunk a SLAM második lépéséhez.

Az adattársítás során az újonnan megfigyelt iránypontokat kell a már meglévőkhöz csatolnunk. Ennek lényege, hogy „folytonosságot" kapjunk a korábbi és az aktuális iránypontok között (mozgásváltozások), ezzel biztosítva a térkép konzisztenciáját. Probléma abból adódhat, ha nem tudunk korábbról azonosított iránypontokat felfedezni vagy ha hibásan társítunk egy már meglévő iránypontho z az aktuális lépésben felfedezettel. Emiatt a robot hibásan becsülheti meg helyét. Az adattársítás a SLAM esetén már nemcsak a lokális ablakokra koncentrálódik, az eljárás globálisan is társítja az iránypontokat globális hurkokat (loop closure) detektálva. Egy-egy hurok azonosításával a korábban megfigyelt értékek pontosíthatók, így az eljárás képes a korábban akkumulált hibát csökkenteni, eltüntetni [35].

Az adattársítás a paradigma egyik kiemelten fontos része. Hiányában vagy nem megfelelő működésében a SLAM vizuális odometriává fajulhat, azzal hogy a globális konzisztencia nem teljesül. A módszer ezen lépése *ismert* iránypontok esetén egyszerűbb, ekkor a globális konzisztencia könnyebben elérhető, hiszen tudjuk mit keressünk és hogy amit keresünk hogyan néz ki. De a SLAM alkalmazható *hasonlóságon* alapuló társítással is, ebben az esetben sokkal bonyolultabb kiválasztani az

iránypontokat (jellemzőket) és azokat egymáshoz társítani.

Utolsó lépésben az aktuális pozíció és a becsült, iránypontok szerinti helyzet összevetése és meghatározása történik, továbbá ha új iránypont került a robot látóterébe az is rögzítésre kerül. Így lesz lépésről lépésre a pozíció és térkép egymást kiegészítve meghatározva, felépítve.



4. ábra. SLAM: becslés és megfigyelés [10]

4.2. Matematikai leírás

Ahogy az a bevezetőben is megjelent, a SLAM egy jelentős bizonytalansággal rendelkező ún. sztochasztikus valószínűségi változókkal jellemzett folyamat, hiszen az összes megfigyelés, helymeghatározás becslés és pozíció megadás a robot kinematikájából és a környezet érzékeléséből pontosan nem, csak valószínűségekben fejezhető ki. Cél, hogy a folyamat ideális esetté konvergáljon és a világ ismételt feltárásával a becsült értékek stabil értékekké váljanak. Probléma megoldására többféle matematikai modell is létezik, ezek közül a leggyakrabban alkalmazott megoldások alapjait jelentő nem-lineáris szűröket és optimalizációs megoldásokat tekintem át [35].

Ahhoz, hogy a SLAM valószínűségi modellje általános módon felírható legyen, használjuk az alábbi jelöléseket:

$_C$ jelölje az egymást követő diszkrét időpontok halmazát

x_t jelölje a t időpillanatban becsült pozíciót (pozíció és orientáció egyaránt), az egymást követő pozíciókat, azaz az útvonalat pedig jelölje $X_C = \{x_0, x_1 ... x_t\}$

u_t jelöle a t időpillanatba vezető szabályozó jelet, a szabályozójelek szekvenciáját pedig az $U_C = \{u_1, u_2 ... ut\}$

m jelölje az elérhető iránypontokat, a tulajdonképpeni térképet

z_t pedig jelölje a t időpillanatban x_t pozícióban megfigyelt iránypontot (iránypontok halmazát), az egymás utáni X_C pontokban megfigyelt iránypontok (iránypont halmazok) sorrendjét pedig a $Z_C = \{z_1, z_2 ... z_t\}$ definiálja

Az imént bevezetett jelölések alapján a SLAM modellje az alábbi összefüggéssel definiálható:

$$P(x_t, m \mid Z_C, U_C) \tag{10}$$

amely szerint az aktuális pozíció becslése és a térkép rögzítése minden egyes t időpillanatban a megfigyelések és a szabályozó jelek feltételes valószínűségén alapul. A megfigyelések és szabályozó jelek a vizsgált idő valamennyi diszkrét időpillanatában mért értékeket és az aktuális időpillanat értéket is tartalmazza, azaz az x_t, m esemény bekövetkezéséhez a teljes időbeli történést figyelembe vesszük (posteriori).

4.2.1. Markov-lánc

Az előzőekben megadott (10) valószínűségi SLAM modell diszkrét sztochasztikus folyamatokat leíró Markov-láncnak is tekinthető, abban az esetben ha a t időpont eloszlását rekurzívan a $t-1$ eloszlásból határozzuk meg. Ezt a Bayes tétel miatt megtehetjük, ehhez azonban ismerni kell a szabályozó jelhez kapcsolódó matematikai modellt, valamint az iránypont megfigyelésére, mérésére vonatkozó egyenletet is [35].

A mozgási modellt (11), ami a robotot a t időpillanatban az u_t jel hatására x_t állapotba visz át x_{t-1}-ből, az alábbiak szerint írhatjuk fel:

$$P(x_t \mid x_{t-1}, u_t) \tag{11}$$

A megfigyelésekre vonatkozó függvény (12) pedig az alábbiak szerint definiálható:

$$P(z_t \mid x_t, m) \tag{12}$$

E két modell alapján a Markov feltétel teljesülése miszerint a rendszer aktuális állapota mellet a jövőbeni állapota nem függ a múltbeliektől könnyen látható. A valósidejű SLAM algoritmusok megoldásai (EKF-SLAM, FastSLAM) ezen az elven működnek.

4.3. Térkép fajták

A SLAM különböző implementációi az alkalmazott érzékelőktől, mérési pontosságuktól és megvalósítástól függően különböző térképeket építenek. Egyrészt beszélhetünk lokális vagy globális térképről, amit az alapján különböztetünk meg, hogy kihez viszonyítva készül a reprezentáció. Másrészt a térképek típusuk szerint is osztályozhatók: ezek szerint 3 kategóriát különíthetünk el: beszélhetünk topológiai, metrikus vagy hibrid térképekről. A képet árnyalja még egy negyedik kategóriai is, a szemantikus térképek köre, de ez többnyire valamely előzőleg felsorolt típushoz kötve, kiegészítésként jelenik meg, így közvetlenül nem kapcsolódik a SLAM térképezéshez, ezért ebben a részben ezt a fajta térképet nem vizsgálom [37][39].

4.3.1. Metrikus térképek

Geometriai térképek építése során a tárgyak metrikus tulajdonságaik (kiterjedése, elhelyezkedése, stb.), geometriai építőelemeik (vonalak, sarkok, pontok) és megjelenésük (színek) szerinti reprezentációja történik. A tárgyak egymáshoz viszonyított helyzetét metrikus módon rögzíti, és mindvégig méretarányos reprezentációra törekszik. Geometriai térképek készülhetnek 2 vagy 3 dimenzióban is, leginkább ismert megoldásai a tulajdonság térképek (feature map) vagy a foglaltsági háló (occupancy grid). A metrikus térképek egyik előnye, hogy az ember által látott módon képes a környezetet ábrázolni (pl.: pontfelhő), segítségével nemcsak útvonaltervezés, -követés hanem pontosabb lokalizáció is megvalósítható, ugyanakkor memóriaigénye jóval nagyobb mint a topológiai térképé.

A foglaltsági háló a környezetet rácsszerű elemekre bontja, ahol minden egyes cellát foglaltsági értékkel fog jellemezni. Ha a cella által leképzett helyen a valós környezetben objektum van, akkor foglaltnak veszi fel (1), más esetben pedig kitöltetlennek (0) jelöli.

4.3.2. Topológikus térképek

A topológiai térképek (topological map) a környezetről alkotott egyfajta magas absztrakciós szintű gráf reprezentációk. A környezet dekompozíciójából alkotott gráf elemei mind-mind jelentéssel bírhatnak a reprezentációban, de alapvetően a csomópontok csak egy-egy kiemelt helyet vagy tárgyat, míg az összekötő élek a közöttük lévő utat, kapcsolatot reprezentálják. A helyek, tárgyak szomszédsági viszonyára fókuszál anélkül, hogy bármilyen metrikus adatot társítana hozzá, így segítségével könnyedén áttekinthető a bejárt terület, útvonal. A topológiai térkép és a szemantikus reprezentáció között van hasonlóság, azonban a kettő közötti különbségre fontos kitérni. Topológia térkép esetén nem kell egzakt módon rögzíteni az adott hely, tárgy típusát, elegendő ha csak azok egyszerűbb megjelölését használjuk pl.: ajtó helyett elérhetetlen terület. A topológiai térkép absztrakt volta miatt könnyedén kezelhető memóriában, azonban leginkább útvonaltervezésre, legrövidebb út keresésre használják, SLAM esetén pedig a globális hurkok detektálására ad hatékony megoldást.

4.4. SLAM kategóriák

A SLAM megoldások kategorizálása igen összetett feladat. A korábbiakból is kitűnik, hogy a paradigma igen szerteágazó, többféle aspektus és szempont szerint kategorizálható, egyértelmű besorolás pedig nem létezik. A felosztás szempontjai lehetnek [39]:

- az alkalmazott szenzorok és leképezési technikák
- az implementációk eltérő matematikai modelljei (szűrő vagy optimalizáló)
- az adattársítás technikái
- vagy a környezet jellege (dinamikus vagy statikus).

Az egyik leggyakrabban alkalmazott kategorizálás szerint kétfajta SLAM-et különböztet meg: teljes (offline) vagy online.

A megértéshez tekintsük az alábbi feltételes valószínűséget (13), amelyben a jelöléseket a 4.2. fejezetben bevezetettek alapján alkalmazom:

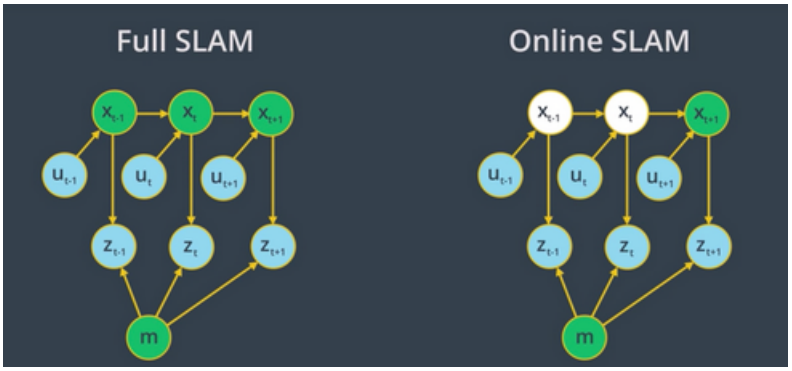
$$P(X_C, m \mid Z_C, U_C) \tag{13}$$

Ebben az esetben már nem az aktuális pozícióra (x_t), hanem a teljes trajektóriára (X_C) vonatkozóan határozzuk meg az eloszlást, azaz minden egyes diszkrét

időpillanatban valamennyi addigi pozíciót (az aktuálisat is beleértve) újrabecslünk a térképpel együtt. Ez elég időigényes és sok erőforrást igénylő feladat, ezért gyakran ezt offline/batch módon végzik, valós időben pedig az online változatot használják.

A 4.2. fejezetben bevezetett (10) feltételes valószínűség az online változatot írja le. Ez a megoldás tulajdonképpen a full SLAM egy részhalmazát jelenti, amely minden egyes időpillanatban csak az aktuális pozíciót becsli a térképpel együtt. A becslés ebben az esetben inkrementálisan végezhető, amely jelentősen csökkenti a számítási kapacitás igényét, így támogatja a valós idejű futtatást.

A SLAM becslési mechanizmusát tekintve beszélhetünk feltétel nélküli (priori) vagy utólagos (posteriori) becslési megoldásról. A kettő abban különbözik egymástól, hogy az aktuális időpillanatban megfigyelt iránypontokat tényként kezeljük-e a becslés meghatározásához vagy sem.



5. ábra. Offline és online SLAM Bayes-hálója

4.5. Vizuális-SLAM

A vizuális-SLAM [37] a vizuális információkat biztosító eszközök használatával kívánja megoldani a vizuális odometria és a SLAM problémáit. Az alkalmazott kameratípusok a vizuális odometriánál bemutatottak lehetnek, de a kutatásokban használnak más típusú szenzorokat (akusztikus szenzorok, lézer távolság mérők), nem csak kifejezetten kamerát.

A vizuális SLAM, ahogy a SLAM is, a globális konzisztenciára törekszik, így a folyamat során a SLAM vizuális odometriát is magába foglaló lépése során kinyert pozíciót teszi globálissá és nemcsak az egymást követő képek alapján határozza meg azt, hanem a korábbi iterációkban kinyert információkat is figyelembe veszi.

A vizuális SLAM architektúrája tipikusan két részre bontható. Beszélhetünk frontend részről és a backend-ről. Előbbi feladata a szenzoroktól jövő adatok modellekbe, transzformációkba való leképezése és a SLAM eljárás során ismertetett adattársítás megoldása.

A backend a transzformációk és a társított adatok alapján többnyire a térképépítés feladatát látja el.

A frontend adattársítás művelete a loop closure (hurokzárás) problémájára is fókuszál. Loop closure probléma akkor áll fent, ha a kamera olyan jelenetet detektál, amit már korábban is rögzített és ez nem szomszédos képeken következik be. A loop closure detektálására többfelé megoldás is létezik. A legkézenfekvőbb a tulajdonságok keresése valamennyi megelőző frame-en, de ez igen költséges lehet. A megoldást sokkal inkább csak a frame-ek egy különleges halmazán végzik el, ezeket kulcs képeknek nevezzük. A kulcs képek kiválasztására leggyakrabban a Bag of Visual Words eljárást alkalmazzák.

5. Rendszerterv

Ahogy azt a diplomamunka bevezetőjében is ismertettem, dolgozatom célja, egy robotrendszer olyan elemének elkészítése, amely az előzőleg bemutatott vizuális odometria és SLAM megoldások segítségével képes a lokalizáció és térképépítés feladatát a legpontosabb módon elvégezni. Ehhez elengedhetetlen a trajektória becslése, valamint a környezet megfelelő rekonstrukciója, amit 3D pontfelhő technológiával kívánok megvalósítani. Jövőbeni céljaim közt szerepel a robotrendszer másik elemének elkészítése is – a szabályozó –, mely a felépített információk alapján lehetővé tenné az autonóm navigációt is, azaz képes lenne szabályozni működését, kontrollálni mozgását, erre azonban diplomamunkám nem terjed ki.

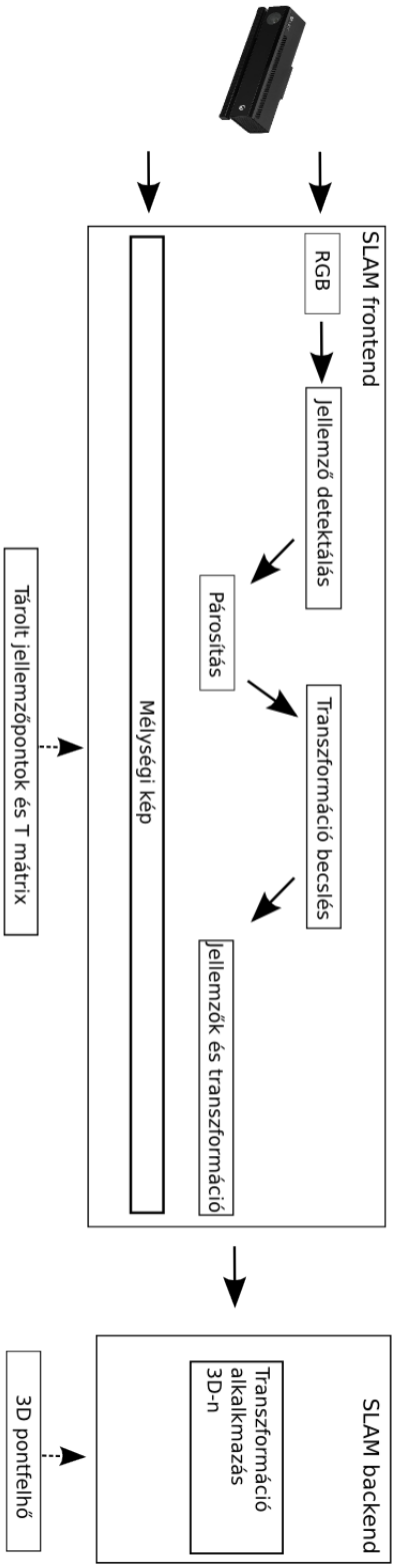
A rendszert a már bemutatott eljárásokra és eszközkészletre alapozva szeretném megvalósítani. A bemutatott kameratípusok közül a feladat megvalósításához a Microsoft RGB-D kamerájának második változatát (Kinect v2) használtam. Ennek oka, hogy fejlesztői oldalról mind a Microsoft, mind pedig az open-source világból megfelelő a támogatottsága (elérhetők a driverek, toolok, libraryk), továbbá rengeteg validációs adathalmaz áll rendelkezésre az eredmények ellenőrzéséhez.

Célom, hogy az eredményt optimalizáció után pontfelhő alapú, 3D térképen reprezentáljam.

Alkalmazott szoftverek:

- ROS – Robot Operating System
- libfreenect2 – Driver Kinect RGB-D kamerához
- iai_kinect – ROS Kinect csomaggyűjtemény
- OpenCV a képműveletek kezeléséhez
- Point Cloud Library a mélységi és RGB adatok alapján történő 3D vizualizációhoz
- Qt alkalmazás framework és wrapper

A következő részben a fentebb felsorolt komponensek bemutatásával foglalkozom. Ezt követően a fejlesztés részletezése, a komponensek integrálása, valamint a telepítés leírása következnek.



6. ábra. Rendszerterv vázlata

6. Felhasznált szoftverek, gyűjtemények bemutatása

Ebben a fejezetben a választott szoftverelemeket kívánom bemutatni, összefoglalva azok jellemző tulajdonságait, leggyakoribb felhasználási területeit, előnyeit és hátrányait. Az áttekintés több keretrendszert és gyűjteményt érint melyek a:

- Robot Operating System (ROS)
- Libfreenect2 driver
- iai_kinect ROS interfész
- OpenCV 2.4
- Point Cloud Library
- PyQt és Qt Desinger

6.1. Robot Operating System

A ROS egy open-source robot operációs rendszer, mely hatékony robotalkalmazások fejlesztését teszi lehetővé. Gyakorlatban a valódi PC-re készült operációs rendszerek és keretrendszerek világa közé helyezhető el - ezért is hivatkoznak rá „meta" operációs rendszerként -, mert önállóan nem futtatható ugyanakkor szoftvergyűjteményeket, és eszközöket (alsó szintű eszközkezelést, hardvertámogatást), stat-of-the-art megoldásokat biztosít, továbbá middlewarehez kapcsolódó funkciót (szinkron és aszinkron üzenetküldés, konfigurációmenedzsment, csomagkezelés) is betölt a robot szenzoraival és aktuátoraival való interakció, valamint a felhasználói alkalmazások kiszolgálása során [30].

Ahogy kezdetben a számítógépek világára is jellemző volt, hogy az alkalmazott architektúrának megfelelő rendszert kapcsoltak az eszközhöz, a robotoknál is ehhez hasonlóan alakult a világ. Minden egyes robot saját rendszerrel, programozói interfésszel és utasításkészlettel rendelkezett holott gyakran ugyanazokat az alapvető műveleteket látták el feladataik végrehajtása során. A ROS egyik kiemelt fókusza pont erre a problémára irányult, azzal, hogy egy olyan programrendszer kifejlesztését tűzték ki célul, ami standard és skálázható módon elfedi a hardver sajátosságokat és egy általános eszközkészletet biztosít minden robot számára, jöjjön az az akadémiai, kutatási területről, az iparból vagy csak az egyszerű érdeklődőtől.

6.1.1. ROS szerkezet

A ROS rendszer felépítését a hálózati architektúrákból ismert hub-ként képzelhetjük el a legegyszerűbben. Erre a hub-ra képesek a robot szenzorai, végrehajtói, a különféle programcsomagok, processzek mint önálló funkcionalitással rendelkező Node-ok csatlakozni és egymással közvetlenül kommunikálni.

A Node-ok végezhetnek számítási feladatokat vagy előállíthatnak, felhasználhatnak mások által biztosított adatokat. Regisztrációjukat és kommunikációs platformjukat a ROS Master látja el, ami a ROS Core része. A ROS Core egyfajta kernel, amely nélkül a kliensek képtelenek lennének kommunikációs folyamatot megkezdeni, vagy szolgáltatásokat nyújtani és igénybe venni. Ezen felül jó néhány közös funkciókat megvalósító szolgáltatást is biztosít kliensei számára, mint például a paraméter management vagy a rosout. A Node-ok készülhetnek többféle nyelven is - több mint 10 programnyelvhez létezik kliens könyvtár -, azonban a köztük lévő kommunikációt ez nem befolyásolja, hiszen annak alapját a kicserélt adat adja.

6.1.2. ROS kommunikációs típsuok

A Node-ok közötti kommunikáció típusa alapján beszélhetünk egyirányú kommunikációt megvalósító publish/subscribe alapú Topic-ról, vagy a kétirányú service hívásról amely esetben a kicserélt üzenetek típusa kérés (request) / válasz (response) alapú lesz. Előbbi esetben minden olyan komponens aki feliratkozott a szolgáltató által publikált és Master által beregisztrált csatornára megkapja az üzenetet anélkül, hogy az üzenet feladója értesülne róla. Request-response esetben a kommunikációt kezdeményező kliens kérésére reagál az adatot szolgáltató szerver, szinkron módon. Bármelyik Node nyújthat ilyen szolgáltatást, regisztrációját azonban ebben az esetben is a Master látja el.

6.1.3. Összefoglalás

A Willow Garage által készített és támogatott ROS alapvető jellemzői összefoglalásként az alábbiak szerint határozható meg: [31]:

- peer-to-peer kapcsolat: a service vagy üzenetküldö szolgáltatáson keresztül mindegyik Node képes a másikkal kapcsolatot tartani szinkron vagy aszinkron módon, akár a robottól távoli fizikai helyen (TCP)
- multi-language támogatás: az üzenetküldő protokoll segítéségével (XML-RPC) a Node-ok működése összehangolható, a közös interfész és alkalmazott proto-

koll lehetővé teszi hogy az egyes programcsomagok más nyelven készüljenek (leginkább azonban Python és C++ használata terjedt el)

- microkernel alapok: sok végrehajtható utasítás adja a rendszer alapját amit a komponensek használhatnak, végrehajtásuk párhuzamos módon történik.
- újrafelhasználás és integrálhatóság mely lehetővé teszi hogy a már létező programcsomagokat ROS interfésszel ellátva egyszerűen felhasználhassuk

A ROS bár eredetileg az ipart is célba vette, néhány biztonsági hiányossága, valós idejű alkalmazhatósága miatt kevéssé terjedt el. Az utóbbi évek eredménye, hogy elkezdődött a keretrendszer újragondolása és így a ROS 2 tervezése és fejlesztése, amely valódi cross-platform megoldásként kívánja a robotikai megoldások világát szolgálni.

6.2. Libfreenect2

A Libfreenect2 egy open-source eszközkezelő, cross-platform driver csomag, amit kifejezetten a Microsoft Kinect második generációs kamerájához készítettek. (A Kinect első generációja a Libfreenect csomaggal kezelhető.) C++ nyelven írták és a mai napig frissítésekkel látja el az OpenKinect community [21].

Az illesztőprogram segítségével, mely kezeli az eszköz által elvárt USB 3 portot, lehetőség van az RGB és mélységi nyersadatok párhuzamos kiolvasására, valamint API-a biztosítja a kamera kalibrációs eljárás végrehajtását is.

6.3. iai_kinect package

Az iai_kinect a brémai egyetem Mesterséges Intelligencia Intézete által fejlesztett csomag- és könyvtárkészlet, mely a libfreenect2 driver által biztosított natív adatokat ROS kompatibilis adathalmazzá transzformálja. Egyfajta hidat képez a driver és a ROS között. A csomag kifejezetten a Kinect v2-höz készült, legfőbb feladata pedig a kamerától érkező nyers adatok ROS topicokba történő továbbítása.

A csomag 5 részből áll:

- a Kinect v2 eszközhöz fejlesztett ROS interfészből
- a driver és ROS közötti adattovábbítást biztosító bridge komponensből
- OpenCV alapú kalibrációs toolból

- OpenCL alapú mélységi regisztrációs toolból
- egyszerű pontfelhő- és/vagy színeskép-megjelenítőből.

Az interfész az alábbi topicokat definiálja, elérésük a következő módon történhet:

$$/kinect2/[felbontás]/image\_[típus](/[formátum]) \tag{14}$$

ahol a felbontás: sd, qhd, hd; a típus: ir, depth, color vagy mono; míg a formátum opcionálisan tömörített is lehet.

Az iai_kinect csomag integrálva van a ROS image_dept_proc csomagjával, így lehetőségünk van a szenzortól jövő adatokat közvetlenül pontfelhő formátumba kiolvasni a $/kinect2/[felbontás]/points$ topic-on keresztül.

A csomag ROS Hydro és Indigo változatokra érhető el, a kezdeti verziók után optimalizálták a feldolgozást és a hálózati adattovábbítást, amely lehetővé tette, hogy az érzékelőtől kapott információkat 30 Hz sebességgel dolgozzák fel. A Kinect szenzor gyártásának és támogatásának megszüntetésével azonban a csomag fejlesztése is abbamaradt. Ennek negatív következménye, hogy a jelenleg is elérhető verzió csak OpenCV 2.4.x verzióval kompatibilis.

6.4. OpenCV 2.4

Az OpenCV (Open Source Computer Vision Library) egy nyílt forráskodú függvénykönyvtár (kezdetben az Intel által támogatott projekt volt), mely valódi multiplatform elven számos operációs rendszeren használható. A gépi látás és a mesterséges intelligencia kutatásainak támogatására helyezi a hangsúlyt hatékony és széles algoritmus megvalósításaival.

Több mint 2500 algoritmust valósít meg a könyvtárgyűjtemény, a képfeldolgozástól kezdve az data mining területén át a gépi látásig. Interfészei segítségével Javaból, C++-ból és Pythonból is használható.

A cv_bridge-nek köszönhetően integrált a ROS-al, ami lehetővé teszi, hogy a ROS üzeneteket OpenCV formátummá konvertáljuk későbbi feldolgozás céljából.

6.5. PyQt

A PyQt az egyik legismertebb Python interface a Qt alkamazás framework-höz. A Qt a Riverban Computing Limited által fejlesztett C++ alapú, széles platformtámogatással rendelkező GUI fejlesztő keretrendszere es eszköztára. Támogatást ad

alacsony szintű hálózati kommunikációra, socketek kezelésére; párhuzamos és többszálú alkalmazások fejlesztésére; adatbázis elérésre; file és könyvtárkezelésre; XML dokumentumok értelmezésére, grafikus komponensek használatára; SVG objektum kezelésre, beépített web browser támogatásra; OpenGL alapú 3D renderelésre.

A Qt előnyét az objektum-orientáltság, valamint azok lazán csatolt mivolta adja. A komponensek/objektumok - legyenek akár más thread-ben - úgynevezett signal/slot mechanizmuson keresztül kommunikálhatnak. Ezek egyfajta eseményvezérelt működést jelentenek, mely eseményekre több objektum is válaszolhat feltéve, ha közös nyelvet beszélnek, azaz szignatúrájuk megegyezik. Ennek előnye, hogy szervezésük nem befolyásolja működésüket.

A Qt Designer pedig grafikus módon ad lehetőséget felhasználói felületek gyors es hatékony tervezéséhez és generálásához.

A Qt előnyeit a Python egyszerűsége teszi meg kézzelfoghatóbbá. A létező C++ alapú library-k modulokká csomagolásával (wrapping) még komplexebb alkalmazások fejlesztését valósíthatjuk meg az interpretált nyelv egyszerűségével. A PyQt megalkotásánál is ez volt a legnagyobb motiváció.

A PyQt több verzióban is elérhető: a PyQt4 a Qt v4-es es v5-ös verzióját támogatja, a PyQt5 a Qt v5-ös, míg a 2021-ben megjelent PyQt 6 a Qt legutolsó kiadásával kompatibilis. Verziótól függően több száz osztályt tartalmaznak modulokba szervezve a Qt által nyújtott funkciók lehető legteljesebb támogatására. Előbbi verziók Python 2.x es 3.x verzióján is támogatottak utóbbi azonban már csak Python 3 platformon használható.

6.6. Point Cloud Library

A Point Cloud Library (PCL) egy nyílt forráskódú, 2 és 3 dimenziós kép és pontfelhő feldolgozást támogató könyvtárgyűjtemény. Széles körben használják lévén, hogy valódi cross-platform funkcionalitással rendelkezik. Elérhető Windows, Linux Android, iOS es MacOS rendszereken egyaránt.

A könyvtár számos algoritmust kínál szűrésre, szegmentációra, regisztrációra, illesztésre stb., melyeket moduláris módon csomagolnak, így az egyes funkciók egymástól függetlenül fordíthatók és telepíthetők és biztosítják, hogy valós 3D képet alkossunk a környezetünkről. Natívan támogatja az OpenNI protokollt, így a pontfelhők mélységi kameráktól történő közvetlen lekérdezését is lehetővé teszi.

A PCL teljeskörűen integrált a ROS-ban, rengeteg funkciót biztosít a vizualizációtól kezdve a PCL library saját fájltípusának (PCD) konvertálásán át, a meg-

jelenítéséig.

A PCLVisualiser osztály a vizualizációért felelős, melynek implementálását VTK alapon valósították meg.

7. Tervezés és fejlesztés dokumentálása

A megvalósítani kívánt alkalmazással szemben támasztott követelményeket - rendszerspecifikációt - az alábbiak szerint határoztam meg:

- a rendszer integrált a ROS-ba
- rendelkezik felhasználói felülettel
- képes a kézben tartott szenzortól származó adatokat feldolgozni és
- a pontfelhőt megjeleníteni

A tervezés és fejlesztés folyamatát jelentősen befolyásolja az elérhető fejlesztői környezet és keretrendszer, így nulladik lépésként - még a tervezést megelőzve - kialakításra került az infrastruktúra. Ez a ROS és egyéb alkalmazások, könyvtárak telepítésében, valamint a RGB-D kamera kalibrációjában merült ki.

7.1. Infrastruktúra

A környezet kialakítása a ROS keretrendszer telepítésével kezdődött. Ezt követte a Kinect driver, majd az iai csomag, végül pedig a képfeldolgozáshoz és megjelenítéshez használt könyvtárak kerültek telepítésre. A végrehajtott utasítások egymást követő lépései részletesen az "A" függelékben olvashatók.

7.2. Kalibráció

Ahogy arról már korábban is szó volt, a jelenetek 3D-s rekonstrukciójához elkerülhetetlen az alkalmazott kamerarendszerek típusától függő kalibrálási folyamat. Ennek eredménye, pontossága ugyanis jelentősen befolyásolja a pontfelhő elemeinek hatékony meghatározását és azonosítását. A kalibrálás során a kamerapárok belső (intrinsic) illetve külső (extrinsic) paramétereinek matematikai leírása történik: a valós világhoz képest kamera pozíció és orientáció, valamint a kamera paraméterei úgymint főpont (camera center), fókusztávolság (focal length) és lencse torzítás (distortion). Kamerarendszerek esetén az egymástól való távolságukat is figyelembe kell venni annak érdekében, hogy a szinkron módon készített (színes és infrared) képek a lehető legjobban illeszkedjenek egymásra és így a mélységi adatok könnyen kalkulálhatók legyenek.

A Kinect One, és így általánosan az RGB-D kamerák kalibrációjához többféle eljárás alkalmazható. Implementálhatunk saját megoldásokat pl.: OpenCV függvénykönyvtár segítségével, de használhatunk már meglévő toolkiteket is, mint pl.: a Matlab Camera Calibration Toolbox-ot. A folyamat végrehajtásához én az iai_kinect ROS csomag kalibrációs tool-ját használtam, ami az OpenCV kalibrálási algoritmusain alapul.

7.2.1. Kalibráció menete

Első lépésben a színes és az infra kamera belső paramétereit határoztam meg két önálló eljárással. Mindkét lefutásnál a 5x7-es sakktáblát használtam, különböző távolságokból és szögekből tekintve. Kb. 25-60 képet készítettem, ezek alapján került meghatározásra a fókusztávolság (f_x, f_y) és a főpont (c_x, c_y) mindkét kameratípusra vonatkozóan.

A torzítási együtthatók meghatározása szintén ebben a lépésben történt, ami megadta a lencsék forgatási és eloltási vektorait (r_1, r_2, r_3, t_1, t_2).

Második lépésben a két kamera külső paramétereinek meghatározását végeztem el azonos homográfia céljából. Az eljárás lehetővé teszi, hogy a mélységi kép pontjait a színes kép koordinátáiba transzformáljuk. Ennek eredményeként a szinkron mód készített színes- és mélységi képpixelek egybeesnek.

A 4. táblázat a kapott külső és belső paramétereket összegzi, a (15) a homogén transzformációs mátrixot mutatja.

A 7. ábra a kameraképek kalibráció előtti és utáni állapotát, valamint az eljárás egy tipikus beállítását mutatja.

7.2.2. Nehézségek

A kalibrációs eljárás időigényes és nehéz feladat, nagyfokú precizitást igényel, sikerét a külső környezeti hatások is jelentősen befolyásolják. A legpontosabb eljárás csak többszöri végrehajtás után sikerült, hol a fényhatások, hol a térben elhelyezkedő diverz objektumok, hol pedig maga az implementáció gátolt a sikerben.

Itt érdemes megjegyezni azt is, hogy a két kamera azonos homográfia céljából történő regisztrálása különös figyelmet igényel, ugyanis a color és depth képek valójában nem szinkron módon készülnek. Ha az alkalmazott sakktábla picit is elmozdul a két kép rögzítése között, a visszavetési hiba igen nagy lehet.

Összességében elmondható, hogy a kalibrálás során 0.2 körüli visszavetési hibát

7. ábra. Kalibráció előtt (felső sor), után (középső sor), tipikus elrendezés (alsó sor)

sikerült elérnem, valamennyi kameratípus esetén.

Kamera	f_x	f_x	c_x	c_y
RGB	1057.73	1067.20	956.03	554.29
IR	357.56	357.72	258.06	201.22

Kamera	r_1	r_2	r_3	t_1	t_2
RGB	.0791	-.1458	.0008	.0037	.0704
IR	.0840	-.0256	-.0010	.0013	.0913

4. táblázat. RGB és IR kamerák kalibrációs paraméterei

$$H_{depth2Color} = \begin{bmatrix} 1.0000 & -.0009 & -.0015 & -.0589 \\ .0009 & 1.000 & -.0127 & -.0118 \\ .0015 & .0127 & 1.000 & .0014 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{15}$$

7.3. Felhasználói kezelőfelület

A rendszer tervezését és fejlesztését a kezelőfelület funkcióinak és követelményeinek meghatározásával kezdtem meg.

7.3.1. UI

A felhasználói felületet struktúráját és elemeinek rendszerét a 8. ábra szerint határoztam meg.

8. ábra. A tervezett felhasználói felület

A megjelenítést tekintve, eszerint két nagy részre lehet bontani a user interfészt. Az alkalmazás bal oldalán az opciók és beállítások paraméterezhetők, valamint itt kaptak helyet a feldolgozást vezérlő gombok és egy a beállításokat, interakciókat, feldolgozást és esetleges hibákat részletező naplózási ablak is. Jobb oldalon a feldolgozott kameraképek alapján elért eredmények jelennek meg: aktuális kamerakép, detektált jellemzőkkel bővített kép, valamint a konstruált pontfelhő.

A felhasználói felület részét képezi még a bal felső sarokban található menü és a jobb alsó területen megjelenő státuszbár is. Előbbi a funkciók gyors elérését, utóbbi pedig a verzió és egyéb termékjellemzők kijelzését biztosítja.

7.3.2. Funkciók

A lehetséges funkciókat tekintve a felhasználó az alábbi aktivitásokat hajthatja végre:

- képek rögzítése
- feldolgozás indítása

- paraméterek állítása
- feldolgozás megszakítása
- pontfelhő mentése
- kilépés

A képek rögzítése a *Recording* gomb megnyomására indul el. A letárolt adatok a későbbi feldolgozáshoz kellenek, így ebben a lépésben bizonyos előfeldolgozási feladatok már megtörténnek. A rögzítést a *Stop recording* gombra kattintva lehet megszakítani.

A tényleges feldolgozás megkezdéséhez a *Start processing* gomb szolgál, ami a korábban rögzített képeket elemzi. A folyamat megszakítására az *Stop processing* gomb használható. A *Start processing* hatására megjelenik az aktuális és a jellemzőkkel tarkított kép, továbbá ha megfelelő kinyert információ áll rendelkezésre, a pontfelhő is elérhetővé válik.

A paraméterek állítása (*Options*) opció lehetővé teszi, hogy a feldolgozás algoritmusát finomhangoljuk. Eszerint lehetőség van a trajektória követés ki- és bekapcsolására, valamint a jellemző detektálási módszerek megválasztására. Fontos követelmény, hogy a paraméterek állítására csak addig van lehetőség, ameddig a feldolgozás el nem kezdődött. Amennyiben azt már elindították, úgy a paraméterek már nem állíthatók, módosításukhoz az aktuális folyamat megszakítására van szükség.

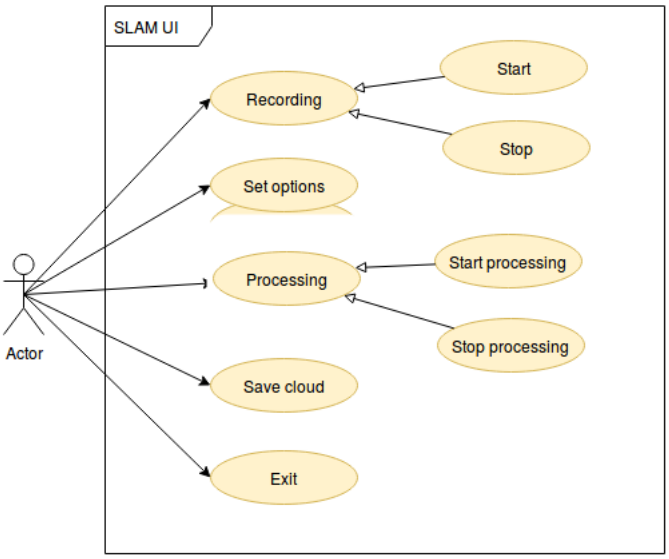
A pontfelhő mentése (*Save cloud*) a feldolgozási folyamat eredményének exportjára szolgáló funkció. Olyan formátumot kell biztosítani, amit a legtöbb alkalmazás képes betölteni és megjeleníteni. A pontfelhő mentése funkció a feldolgozás leállítását követően jelenik meg és mindaddig van rá lehetőség, ameddig új folyamat nem kerül indításra.

A kilépés (*Exit*)funkció az alkalmazás valamennyi erőforrásának felszabadításával és az ablak bezárásával járó, visszavonhatatlan esemény.

A követelményben definiált használati eseteket a 8. ábra mutatja.

7.4. Architektúra

Az architektúra kialakításánál a felhasználói felület és a megvalósítandó térképépítő eljárás kettéválasztására törekedtem. A közös util függvények kivételével az oszályokat tekintve jól elkülönül a két réteg egymástól. Követve a sztenderdeket,



9. ábra. A használati esetek diagramja

kialakításra került a grafikus felület (frontend), valamint az adatok feldolgozását végrehajtó ROS csomag (backend). A kettő közötti kommunikációt a ROS által biztosított kommunikációs architektúra (topic és request/response) segítségével valósítottam meg.

7.4.1. Frontend

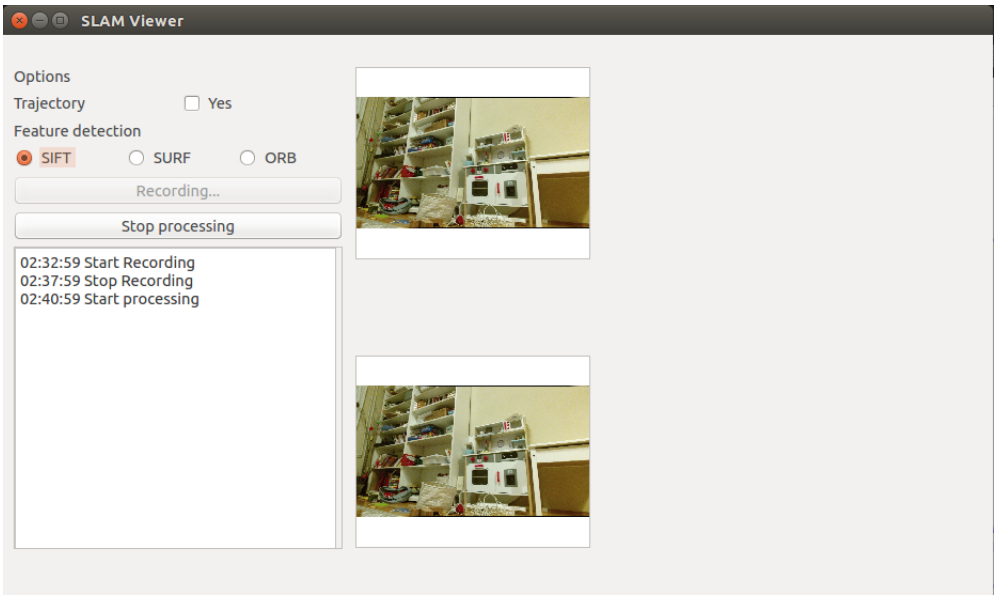
A grafikus interfészt (10. ábra) Python nyelven, PyQt 4 alapon valósítottam meg. Ennek oka, hogy egyszerű designer toolok segítségével összekattintható a kívánt felület, így a figyelmet a funkciók implementálására lehet fordítani.

A felhasználói felület az előzetes designterv szerint került kialakításra. Baloldalon kaptak helyet a paraméterválasztó és funkciógombok, míg jobboldalon a feldolgozás állapotát megjelenítő komponensek kerültek elhelyezésre. A PyQt és a használt PCL Python wrapper integrálhatóságánál azonban nem várt problémába ütköztem. A két verzió nem kompatibilis egymással, így a közös ablakban történő pontfelhő megjelenítés nem támogatott. PyQT 4 alapon a PCL Visualizer widgetbe való integrálhatósága nem lehetséges. Emiatt a felhasználói felület *Start processing* funkciójára való kattintással egy újabb ablak nyílik meg, ami a pontfelhő megjelenítését végzi.

A feldolgozást befolyásoló paraméterek értékeinek tárolása a ROS által nyújtott paraméter szerver segítségével történt meg, így a felhasználó által eszközölt változások egyszerűen letárolhatók és a backend tudomására hozhatók.

Két konfigurációs paraméter értelmezett:

- trajctorie (*bool*)
- feature_type (*string SIFT,SURF,ORB*)



10. ábra. PyQT 4 alapon készített alkalmazás felülete

7.4.2. Backend

A backend architektúráját a ROS architektúrájának megfelelően építettem fel. A teljes backend eljárást Python nyelven terveztem lefejleszteni, de ahogy már a frontend-nél is, itt is problémába ütköztem a PCL library használatával. A wrapper osztály számos PCL funkciót nem támogat (pl. pontfelhő transzformációt), így a backend egyes részeit C++ nyelven írtam meg, hogy elérjem a PCL teljes funkciókészletét. A ROS terminológiájában ez nem jelent problémát, mert a node-ok közti kommunikációt csak a definiált interfészek és protokollok határozzák meg, a node fejlesztéséhez használt nyelv nem befolyásolja a felhasználhatóságot, működést.

A backend ennek megfelelően 3 node-ra bontható, amely csomagszervezést tekintve a `master_thesis` névtérbe került a frontend-el közösen. Így a `master_thesis` package az alábbi modulokat tartalmazza:

- frontend alkalmazás integrációjáért felelős node (Python)
- a képrögzítésért felelős backend service node (Python)

- vizuális odometira és SLAM eljárást implementáló osztályok és utilok összessége (Python)
- a pontfelhő transzformációjáért felelős komponens (C++).

A `master_thesis` csomag valamennyi komponensének elindulási feltétele, hogy az `iai_kinect` csomag `kinect_bridge` nevű node-ja elérhető legyen és az általa biztosított ROS topicok láthatóak legyenek.

A nodek indítása a következő parancsokkal történik:

- `rosrun master_thesis application.py`
- `rosrun master_thesis image_recorder.py`
- `rosrun master_thesis pcl_transformer`

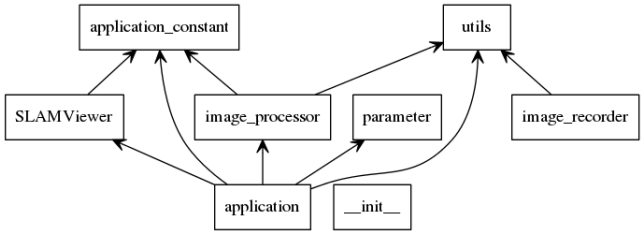
11. ábra. A saját és felhasznált ROS csomagok listája rqt_graph segítségével vizualizálva

7.5. Implementáció részletei

7.5.1. Osztályok és funkcióik

A 12. ábra a megvalósított osztályokat és a közöttük lévő kapcsolatokat mutatja. A legfontosabb osztályok attribútumaikkal és metódusaikkal együtt a 13. ábrán

láthatóak.



12. ábra. A frontend es backend osztályok hierarchiája

7.5.2. Eljárás részei

Az implementált alkalmazás legfontosabb metódusai a képek rögzítésével, valamint a képek feldolgozásával kapcsolatosak. Ezek megvalósításait az alábbiakban kívánom bemutatni.

Kép rögzítés, előkészítés

A kép rögzítése során a `kinect_bridge` által biztosított topic-okból kerül kiolvasásra az adat szinkron módon. A szinkronitást a `message_filters` könyvtár biztosítja. A használt `ApproximateTimeSynchronizer`, lényege, hogy a különböző forrásokból származó adatokat csak akkor bocsátja rendelkezésre, ha valamennyi bejövő adat azonos időpecséttel megérkezett.

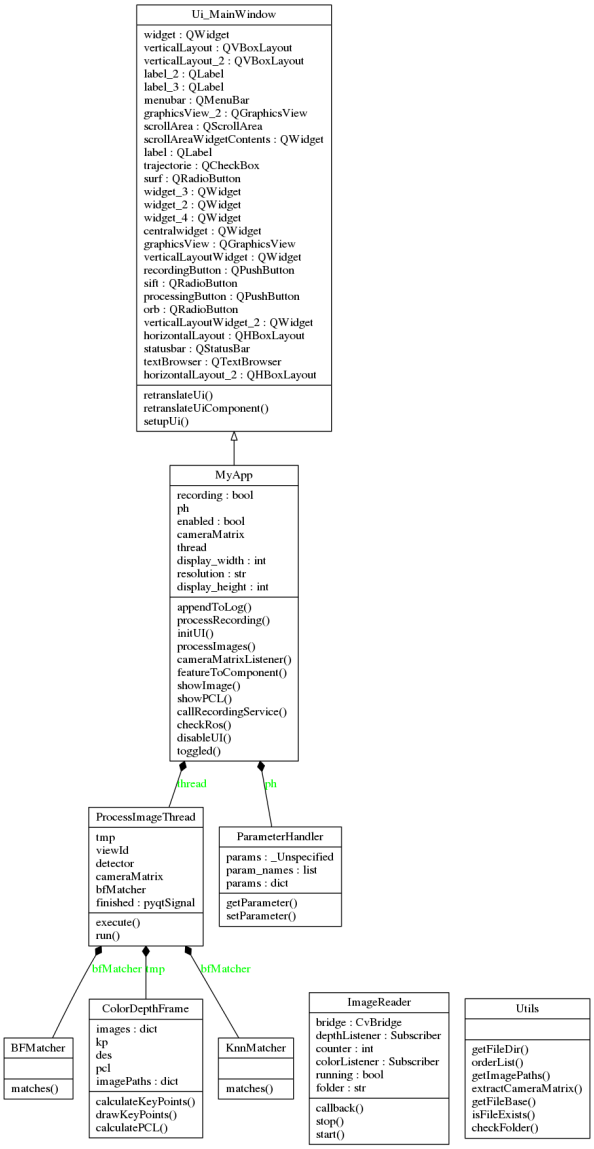
A használt két topic a

- `/kinect2/qhd/image_color_rect` és a
- `/kinect2/qhd/image_depth_rect`.

A mélységi kép regisztrálva érkezik, a rotációs és transzlációs adatok alapján a re-projekciót a `bridge` elvégzi, így a művelet végét a színes és mélységi képek letárolása jelenti, előbbit BGR, utóbbit szürkeárnyalatos színtérben kódolva.

Jellemzők detektálása és párosítása

Amennyiben a képek rögzítésre kerültek és elindul a feldolgozás, az eljárás első lépésben a `feature_type` konfigurációs paraméternek megfelelően a képen található jellemzőket keresi. A jellemező keresés az OpenCV SIFT, ORB és SURF detektorai segítségével történik. A megtalált jellemzőkkel módosított képet megjeleníti



13. ábra. A frontend es backend osztályok vázlatos diagramja

az alkalmazás, majd amennyiben további kép áll rendelkezésre, azon is végrehajtja a jellemző pontok keresését.

A detektálást követően - a második képtől kezdve - történik két egymást követő frame közös jellemzőinek párosítása, melyhez az OpenCV `BFMatcher` objektumát használtam. A brute-force párosító egyszerűen a két kép detektált jellemzőit várja paraméterként. Működése során az első halmaz egy adott jellemzőjét veti össze a második halmaz valamennyi elemével és egy távolságmetrika (pl.: Euklédeszi, Hamming, SSD) alapján a legközelebbi értékű jellemzőt választja megfelelő párnak.

SIFT és SURF descriptorok esetén az eredményt tovább kell szűrni, mert az algoritmus az adott ponthoz a két leginkább hasonlító (legközelebbi) pontot adja

vissza. A pontot abban az esetben lehet figyelembe vennünk, ha a két legközelebbi pont egymáshoz viszonyított aránya kisebb egy küszöbértéknél.

Perspektív-n-pont (PnP) transzformáció

PnP problémáról akkor beszélhetünk, amikor egy kalibrált kamera pozíciójának és orientációjának a meghatározását szeretnénk megadni térbeli referencia pontok és az ezeknek megfelelő 2D pixelpontok segítségével. A megoldáshoz az előző lépésben megtalált jellemzőpárokat használjuk fel. A mélységi kép alapján minden jellemzőpárra meghatározzuk a térbeli pontot és a hozzá tartozó 2D pixelpontot, majd RANSAC eljárással meghatározzuk a rotációs és transzlációs vektorokat. Az elmozdulás becsléséhez az OpenCV `solvePnPRansac` metódust használtam, nagy számú iterációval.

Pontfelhő készítés

Pontfelhő készítése regisztrált mélységi kép segítségével történhet. Mivel a `kinect_bridge` regisztrált mélységi képet ad vissza, az eljárás során az RGB pixeleket, a mélységi adatokat $POINT(X, Y, Z)$ koordinátává kell transzformálni a kameramátrix alapján. A transzformáció tulajdonképpen a koordináta-rendszer átalakítását jelenti: a képtér koordináta-rendszere átalakul világ koordináta-rendszerré. A konverzió során mindazokat a pixeleket, amelyek nem szolgáltatnak megfelelő mélységi adatot kiszűrjük, ezzel is gyorsítva az eljárást.

A 3D-s pont képsíkra való $(u, v) = f(X, Y, Z)$ leképezése az alábbi matematikai összefüggéssel írható le:

$$p = K[R|t] * P \tag{16}$$

amely tovább bontva:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{17}$$

ahol az

- s skálafaktor

- p a leképezett pont a képtérben
- K a kamera belső tulajdonsága (főpont, fókusztávolság)
- $[R|t]$ a kamera külső paraméterei amely a 3D pontnak a kamerához viszonyított relatív transzformációját mutatja
- P 3D pont az euklideszi térben

Inverz esetben, mivel a $P(X, Y, Z)$ pont a világ koordináta-rendszerben van, az $[R|t]$ mátrixot figyelmen kívül hagyhatjuk. Ebből a (17) egyenletet a következőképpen fejthetjük tovább:

$$u = \frac{1}{s_x} f \frac{X}{Z} + c_x \tag{18}$$

és

$$v = \frac{1}{s_y} f \frac{Y}{Z} + c_y \tag{19}$$

Mivel a mélységi képből a Z érték rendelkezésre áll, a X és Y megadható. A pontfelhő fenti egyenletnek megfelelő konstruálási eljárását az alábbi pszeudokód mutatja:

```
FUNCTION COBVERT2DTO3D(CAM_MATRIX, COLOR, DEPTH, SCALE):
      fx,fy <- CAM_MATRIX.fx,CAM_MATRIX.fy
      cx,cy <- CAM_MATRIX.cx,CAM_MATRIX.cy
      WIDTH <- DEPTH.width
      HEIGHT <- DEPTH.height
      POINTS <- []
      FOR v in range(0, HEIGHT):
            FOR u in range(0, WIDTH):
                  Z <- DEPTH[v][u] / SCALE
                  OUTPUT Z
                  IF Z = 0 THEN
                        continue
                  END
                  X <- (u - cx) * Z / fx
                  Y <- (v - cy) * Z / fy
                  C <- COLOR_VALUE(COLOR[v][u])
                  APPEND(POINTS, [X, Y, Z, C])
```

```
            END
        END
        RETURN POINTS
END
```

Transzformáció végrehajtás, illesztés

Az eljárás egyik legfontosabb lépése a transzformáció végrehajtása. Ekkor a korábbi iterációk során konstruált pontfelhőhöz illesztjük az aktuális pontfelhőt az előző lépésben kapott transzformációs mátrix alapján. A pontfelhő-regisztráció tulajdonképpen a relatív póz meghatározását jelenti az éppen feldolgozás alatt álló és a meglévő pontfelhő között. Az egymást követő relatív pózok tárolásával kapjuk a trajektóriát. A pontfelhő illesztése a `pcl_transformer` node segítségével történik (C++ nyelven írt), mely bemenő paraméterként a jelenlegi és új pontfelhőt, valamint a köztük lévő transzformációt kapja. A végrehajtott illesztés eredménye a pontfelhő ablakban láthatóvá válik.

8. Eredmények

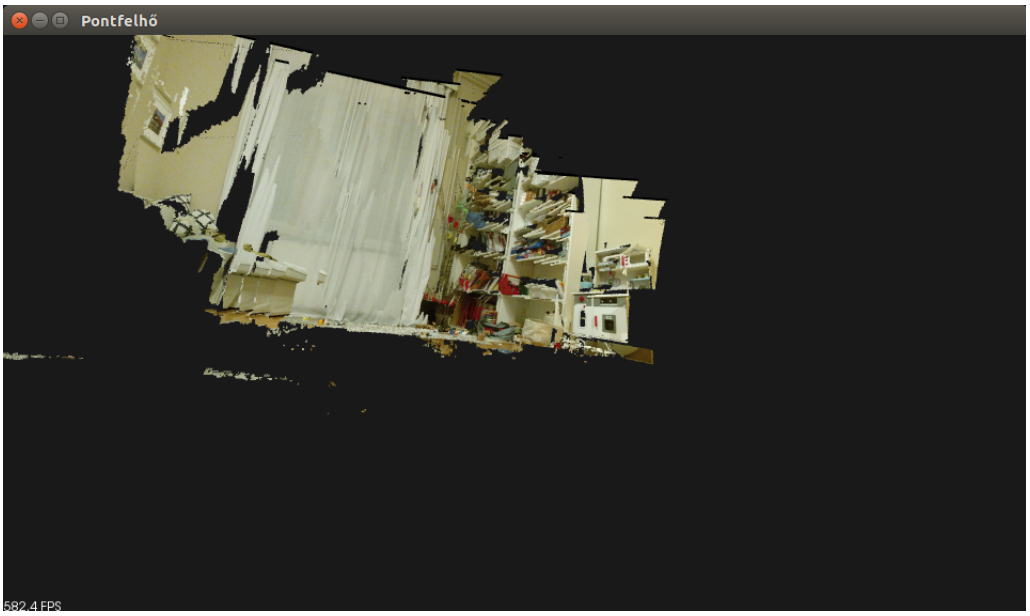
Az eredmények kiértékelése ugyanazon az adathalmazon történt. A képek jó megvilágítás mellett, belső térben kerültek rögzítésre kalibrált kamerát használva. A kamera állványon került elhelyezésre, de a rögzítés során mindvégig kézben tartva maradt.

Általános eredményként elmondható, hogy az implementáció SIFT jellemző detektor használatával adta a legjobb pontfelhő megjelenítést. Ebben az esetben azok a kritikus pontok kerültek megtalálásra és párosításra, amelyek a transzformációs mátrix pontos meghatározásához vezettek.

SURF esetén szintén stabil pontfelhőt hozott létre az eljárás, de itt már néhány fals jellemző is párosításra került, így a transzformáció meghatározása pontatlanabb értékkel került meghatározásra.

ORB esetén már olyan jellemzők is párosításra kerültek két egymást követő képen, amelyek teljesen hibásak voltak, így az eredményt negatív irányba sodorták. A párosítást követően sok esetben nem elegendő számú jellemzőpont maradt, így a transzformációs mátrix meghatározása sem volt lehetséges.

Bár eredményeim nem feleltek meg maradéktalanul előzetes elvárásaimnak, a fejlesztés során számos olyan kihívást oldottam meg és kisebb részeredményekre jutottam, amik későbbiekben további fejlesztések kiindulópontjai lehetnek.



14. ábra. SIFT detektor használatával előállított pontfelhő részlet

9. Továbbfejlesztési lehetőségek

A rendszer továbbfejlesztésének lehetőségét 3 nagyobb kategóriába tudnám osztani. Ezek mindegyike az alkalmazásra és annak használhatóságának javítására fókuszálnak. Megvalósíthatók külön-külön is, de akár iterációban, egymást követő lépésekben is.

A kategóriák az alábbiak szerint alakulnak:

- az alkalmazás SLAM eljárásnak való teljes megfelelése és finomhangolása
- keretrendszer és könyvtárcsomagok frissítése
- az alkalmazás szabályozó rendszerrel való kiegészítése

9.1. SLAM eljárás pontosítása

A lefejlesztett alkalmazás leginkább a vizuális odometria algoritmusának lépéseit implementálja. Pontosabb és konzisztensebb térkép építhető ha a SLAM kulcselemei is megvalósításra kerülnek benne. A továbbfejlesztés egyik legfontosabb lépésének ezt tartanám. Felkészíteni az algoritmust

- a kulcs képek meghatározására,
- gráfoptimalizációval végrehajtott pozíció követésre, valamint
- a globális és lokális loop-closure-k detektálására.

Emellett finomítani lehet a teljes folyamatot, a valós idejű térképépítést és hatékonyságot szem előtt tartva.

9.2. Keretrendszer és könyvátrcsomagok frissítése

A fejlesztés egyik legnagyobb kihívását az adta, hogy bizonyos komponenseknél korábbi verziójú csomagokat kellett használni mind a ROS, mind pedig az iai_kinect csomag kompatibilitása miatt. Bár a használt komponensek számos alapvető problémát, kérdést megoldottak (pl.: rectification), sokszor azonban meg is kötötték a fejlesztés során használható algoritmusok, eljárások lehetőségeit. Ebből kiindulva érdemes lenne egy következő iterációban az alkalmazott komponensek újabb verziójára áttérni (pl.: ROS upgrade, OpenCV upgrade), ami akár a használt RGB-D eszköz leváltását is maga után vonhatja.

9.3. Szabályozó integrálása

A kifejlesztett eljárás hosszú távú célja az autonóm navigáció megvalósítása lehet. Ehhez elengedhetetlen lenne a rendszer valódi robotplatformra való integrációja, a robottól érkező más adatok szenzorfúziója és egy szabályozó (pl. PID) integrálása, mely a bejövő adatok alapján (kamera, odometria, IMU...) egy elvárt referenciához viszonyítva képes lenne szabályozni működését, kontrollálni mozgását.

Összefoglalás

Diplomamunkám témája a lokalizáció és navigáció során történő térképépítő eljárásokra, valamint azok kihívásaira fókuszált. Az elméleti áttekintés mellett ROS alapokon megterveztem és implementáltam egy felhasználói felülettel rendelkező alkalmazást, amely képes a beérkező adatok alapján a környezet rekonstruálására és a térkép megjelenítésére.

A térképépítő megoldások igen sok kategóriába sorolhatók, én a vizuális megoldásokat helyzetem előtérbe, ehhez pedig a Microsoft speciális eszközét, a Kinect v2 RGB-D kameráját használtam.

A tervezés és fejlesztés során elkészült az alkalmazás grafikus felülete, a vizuális odometria és a SLAM megoldásain alapuló eljárás futtatható változata - mely előkészíti és feldolgozza a bejövő adatokat, detektálja a mozgásváltozást, meghatározza a 3D pontfelhőket és konzisztens térképpé építi azokat -, a ROS node-ok kialakítása, és a telepítési dokumentáció összefoglalója.

A megoldás előnyét maga a ROS adja: moduláris, nem kötődik a használt kameratípushoz, egyszerűen integrálható más kamerához, vagy olyan eszközhöz amely mélységi adatokat szolgáltat.

A témaválasztásom célja az volt, hogy olyan területre fókuszáljon, amelynek eredménye sok helyen felhasználható. Az implementáció jelenlegi verziója ugyan pontos térképet épít, a globális konzisztencia hiánya miatt azonban ezt csak lokálisan teszi. Továbbfejlesztendő az eljárás hatékonysága is, a real-time feldolgozás elérése az alkalmazott algoritmusok finomítását igényli.

Az elméleti áttekintéssel és megvalósítással sokat tanultam, tapasztalatot és jártasságot szerezhettem a robotika e szerteágazó világáról.

Summary

The topic of my master thesis focused on map building techniques and their challenges in localisation and navigation. In addition to a theoretical overview, I designed and implemented a user interface application based on ROS that is able to reconstruct the environment based on the incoming data and display the pointcloud map.

Map building solutions fall into many categories, but I focused on visual solutions, using Microsoft's special tool, the Kinect v2 RGB-D camera.

During the design and implementation, I have developed the application's grapchical interface, the executable version of the process built in visual odometry and SLAM standards - that prepares and processes incoming data, detects the motion changes, creates the 3D pointclouds and builds them as consistent map -, the reuqired ROS nodes, and the summary of the installation documentation.

The advantage of the implemented solution is the ROS itself: it highly supports modularity. The solution is not tied to the type of camera used, it can be easily integrated with other cameras or devices that provide depth data.

The aim of my topic was to focus on an area where the results could be used in many places. The current version of the implementation builds an accurate map, but due to the lack of global consistency it does so only locally. The efficiency of the method also needs to be improved, and achieving real-time processing requires refinement of the algorithms used.

Through the theoretical overview and implementation, I have learned a lot, gained experience and expertise in this diverse world of robotics.

Ábrajegyzék

1. A helymeghatározás, az útvonal tervezés és a térképezés kapcsolata. 14
2. A bemutatott RGB-D kamerák [4] [3] [17] 25
3. A VO folyamatának fő lépései 26
4. SLAM: becslés és megfigyelés [10] 40
5. Offline és online SLAM Bayes-hálója 44
6. Rendszerterv vázlata 47
7. Kalibráció előtt (felső sor), után (középső sor), tipikus elrendezés (alsó sor) 56
8. A tervezett felhasználói felület 57
9. A használati esetek diagramja 59
10. PyQT 4 alapon készített alkalmazás felülete 60
11. A saját és felhasznált ROS csomagok listája rqt_graph segítségével vizualizálva 61
12. A frontend es backend osztályok hierarchiája 62
13. A frontend es backend osztályok vázlatos diagramja 63
14. SIFT detektor használatával előállított pontfelhő részlet 67
15. Telepítés ellenőrzése Protonect lib segítségével. Balról jobbra haladva: nyers infra, színes és számított mélységi kép együtt, valamint a nyers színes és számított mélységi külön) 81

Táblajegyzék

1. A helymeghatározás szempontjából csoportosított érzékelők és tulajdonságaik 19
2. Mono, sztereo és RGB-D kamerák jellemzői 24
3. Sobel maszkpár x és y irányú G_x, G_y deriváltak közelítéséhez 30
4. RGB és IR kamerák kalibrációs paraméterei 56

Irodalomjegyzék

[1] 4 Companies Which Stand To Benefit From 3D Sensing,
https://seekingalpha.com/article/4091019-4-companies-which-stand-to-benefit-from-3d-sensing,
utoljára megtekintve: 2020.05.28

[2] Aqel, M. O. A., Marhaban, M. H., M., Saripan I., Napsiah Bt. Ismail: Review of visual odometry: types, approaches, challenges, and applications, SpringerPlus, Vol. 5., 2016 p. 1897

[3] Asus Xtion Pro Live,
https://www.asus.com/3D-Sensor/Xtion_PRO_LIVE/specifications/,
utoljára megtekintve: 2020.05.28

[4] Azure Kinect,
https://docs.microsoft.com/hu-hu/azure/kinect-dk/hardware-specification
utoljára megtekintve: 2020.05.28

[5] Azure Kinect SDK, https://azure.microsoft.com/hu-hu/services/kinect-dk/,
utoljára megtekintve 2021.05.08

[6] Bräunl, T.: Embedded Robotics, ISBN: 978-3-540-70533-8, Springer Berlin, Heidelberg, 2008

[7] Canny Edge Detection Step by Step
https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123,
utoljára megtekintve: 2021.05.08

[8] Chhaniyara, S., Althoefer, K., Seneviratne, L. D.: Visual odometry technique using circular marker identification for motion parameter estimation in Advances in Mobile Robotics, Editor: L. Marques ISBN: 978-981-283-576-5 2008, pp. 1069-1076

[9] Dr. Rövid, A., Dr. Vámossy, Z., Dr. Sergyán, Sz.: A gépi látás és képfeldolgozás párhuzamos modelljei és algoritmusai, Typotex Kiadó, 2014

[10] Durrant-Whyte, H., Bailey, T.: Simultaneous localization and mapping: Part I. in IEEE Robotics and Automation Magazine, Vol. 13., 2006, pp. 99–108.

[11] Durrant-Whyte, H., Bailey, T.: Simultaneous localization and mapping Part II. in IEEE Robotics and Automation Magazine, vol. 13, 2006, pp. 108–117

[12] Fernández-Madrigal, J.-A.: Simultaneous Localization and Mapping for Mobile Robots: Introduction and Methods: Introduction and Methods, ISBN: 978-1-466-62105-3, IGI Global, 2012

[13] Forsyth, D. A., Ponce, J.: Computer Vision (A modern approach), Second edition, ISBN: 978-0-13-608592-8, Pearson by Prentice Hall, USA 2012

[14] Henry, P., Krainin, M., Herbst, E., Ren, X., Fox, D.: RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments, The International Journal of Robotics Research, Vol. 31., 2012 pp. 647-663

[15] Hoffmann, J.: Proprioceptive Motion Modeling for Monte Carlo Localzation in Robocup 2006: Robot Soccer World Cup X, Editors: Gerhard Lakemeyer, Elizabeth Sklar, Domenico G. Sorrenti, Tomoichi Takahashi, ISBN: 978-3-540-74023-0, Springer Berlin Heidelberg, 2007, pp. 258-269

[16] Intel RealSense Compare, `https://www.intelrealsense.com/compare/`, utoljára megtekintve 2021.05.08

[17] Intel RealSense D400, `https://www.intel.com/content/dam/support/us/en/do` `technologies/intel-realsense-technology/Intel-RealSense-D400-Series-Datasheet.pdf`, utoljára megtekintve 2021.05.08

[18] Introduction to Harris Corner Detector `https://medium.com/data-breach/introduction-to-harris-corner-detector-32a88850b3f6`, utoljára megtekintve: 2021.05.08

[19] Introduction to SURF `https://medium.com/data-breach/introduction-to-surf-speeded-up-robust-features-c7396d6e7c4e`, utoljára megtekintve: 2021.05.08

[20] Kim, J., Yoon K.-J., Kweon, I. S.: Robust 3D Visual SLAM in Large-Scale Envrionment in Robotics Research: The 14th International Symposium ISRR,

Editors: Cédric Pradalier, Roland Siegwart, Gerhard Hirzinger, ISBN: 978-3-642-19456-6, Springer Berlin Heidelberg 2011, pp. 519-534

[21] Libfreenect v2
https://github.com/OpenKinect,
utoljára megtekintev: 2021.05.08

[22] Lowe, D. G. Distinctive Image Features from Scale-Invariant Keypoints in International Journal of Computer Vision, Vol. 60., 2004, pp. 91–110.

[23] Nirmala, G., Dr.Geetha, S., Dr.Selvakumar, S.: Mobile Robot Localization and Navigation in Artificial Intelligence: Survey, Computational Methods in Social Sciences, Vol. IV., 2016 pp. 12-22

[24] Nüchter, A.: 3D Robotic Mapping, ISBN: 978-3-540-89883-2, Springer Berlin, Heidelberg, 2009

[25] OpenCV: Harris Corner Detection
https://docs.opencv.org/master/dc/d0d/tutorial_py_features_harris.html,
utoljára megtekintve: 2021.05.08

[26] OpenCV: Intorduction to SIFT
texttthttps://docs.opencv.org/master/da/df5/tutorial_py_sift_intro.html,
utoljára megtekintve: 2021.05.08

[27] Payá, L., Gil, A., and Reinoso, O.: A State-of-the-Art Review on Mapping and Localization of Mobile Robots Using Omnidirectional Vision Sensors, Journal of Sensors, Vol. 2017., 2017 pp.

[28] Poddar, S., Kottath, R., Karar, V.: Evolution of Visual Odometry Techniques

[29] Riisgaard, S., Blas, M.: SLAM for Dummies: A Tutorial Approach to Simultaneous Localization and Mapping
https://dspace.mit.edu/bitstream/handle/1721.1/119149/16-412j-spring-2005/contents/projects/1aslam_blas_repo.pdf,
utoljára megtekintve: 2021.05.08

[30] ROS Introduction
http://wiki.ros.org/ROS/Introduction,
utoljára megtekintve: 2020.05.28

[31] ROS - Robot Operating System
https://blog.generationrobots.com/en/ros-robot-operating-system-2/,
utoljára megtekintve: 2020.05.28

[32] Scaramuzza, D., Fraundorfer, F.: Visual Odometry Tutorial Part I, IEEE Robotics & Automation Magazine Vol. 18., 2011, pp. 80 - 92

[33] Scaramuzza, D., Fraundorfer, F.: Visual Odometry Tutorial Part II, IEEE Robotics & Automation Magazine Vol. 19., 2012, pp. 78 – 90

[34] Self Driving Car localization, https://towardsdatascience.com/self-driving-car-localization-f800d4d8da49,
utoljára megtekintve: 2020.05.28

[35] Siciliano, B., Khatib, O.: Handbook of Robotics, ISBN: 978-3-540-23957-4, Springer Berlin, Heidelberg, 2008

[36] Siegwart, R., Nourbakhsh, I. R., and Scaramuzza, D.: Introduction to Autonomous Mobile Robots, Second Edition, ISBN: 978-0-262-01535-6, The MIT Press, London, 2011

[37] Simultaneous Localization and Mapping (SLAM)
https://www.mathworks.com/discovery/slam.html,
utoljára megtekintve: 2021.05.08

[38] Stachniss, C: Exploration and Mapping with Mobile Robots, University of Freiburg, Department of Computer Science, 2006
http://www.informatik.uni-freiburg.de/~stachnis/pdf/stachniss06phd.
utoljára megtekintve: 2021.05.08

[39] Tutorials: SLAM algorithms
https://www.mrpt.org/List_of_SLAM_algorithms,
utoljára megtekintve: 2021.05.08

[40] Yousif, K., Bab-Hadiashar, A., Hoseinnezhad, R.: An Overview to Visual Odometry and Visual SLAM: Applications to Mobile Robotics, Intelligent Industrial Systems, Vol. 1., 2015 pp. 289-311

Mellékletek

A. Környezet installálása

A.1. ROS Indigo telepítése

Ellenőrizzük a használt Linux disztribúció típusát, verzióját:

```
$ cat /etc/lsb-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=14.04
DISTRIB_CODENAME=trusty
DISTRIB_DESCRIPTION="Ubuntu 14.04.6 LTS"
```

Vegyünk fel az előbbi verzió szerinti ROS repository-t a PPA/third party szoftverforrások közé és töltsük le a megbízható szerző PGP alapú publikus kulcsát:

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

$ sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com
   ↪ :80' --recv-key
   ↪ C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
Executing: gpg --ignore-time-conflict --no-options --no-
   ↪ default-keyring --homedir /tmp/tmp.0at5mQN9iK --no-auto
   ↪ -check-trustdb --trust-model always --keyring /etc/apt/
   ↪ trusted.gpg --primary-keyring /etc/apt/trusted.gpg --
   ↪ keyring /etc/apt/trusted.gpg.d/gezakovacs-pdfocr.gpg --
   ↪ keyring /etc/apt/trusted.gpg.d/mc3man-trusty-media.gpg
   ↪ --keyring /etc/apt/trusted.gpg.d/ubuntu-advantage-esm-
   ↪ infra-trusty.gpg --keyserver hkp://keyserver.ubuntu.com
   ↪ :80 --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
gpg: requesting key AB17C654 from hkp server keyserver.ubuntu
   ↪ .com
gpg: key AB17C654: public key "Open Robotics <
   ↪ info@osrfoundation.org>" imported
gpg: Osszesen feldolgoztam: 1
```

```
gpg: importalva: 1 (RSA: 1)
```

Az újonnan beállított forrásnak megfelelően frissítsük az apt-get gyorsítótárat:

```
$ sudo apt-get update
```

Telepítsük a teljes ROS Indigo-t:

```
$ sudo apt-get install ros-indigo-desktop-full
```

ROSDEP inicializálása:

```
$ sudo rosdep init
Wrote /etc/ros/rosdep/sources.list.d/20-default.list
Recommended: please run

      rosdep update
$ rosdep update
```

Bash módosítása a ROS környezeti változók hozzáadásával:

```
$ cp ~/.bashrc ~/.bashrc_bkp
$ echo "source /opt/ros/indigo/setup.bash" >> ~/.bashrc
$ source ~/.bashrc
```

Catkin (CMake alapú) munkaterület létrehozása és Bash módosítása:

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
$ catkin_make

$ echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
$ source ~/.bashrc
```

A.2. Libfreenect2 driver installálása

Letöltés, build toolok telepítése:

```
$ cd ~
$ git clone https://github.com/OpenKinect/libfreenect2.git

$ cd libfreenect2/
$ cd depends
$ ./download_debs_trusty.sh
$ sudo apt-get install build-essential cmake pkg-config
```

OpenGL és USB támogatás az előzően letöltött debian csomagok alapján:

```
$ sudo dpkg -i debs/libusb*deb
$ sudo apt-get install libturbojpeg libjpeg-turbo8-dev
$ sudo dpkg -i debs/libglfw3*deb
$ sudo apt-get install -f
$ cd ..
$ mkdir build && cd build
$ cmake .. -DCMAKE_INSTALL_PREFIX=$HOME/freenect2
$ make
$ make install
```

A szenzorhoz való hozzáférés megadása a Dinamikus eszközkelezőn (udev) keresztül:

```
$ sudo cp ../platform/linux/udev/90-kinect2.rules /etc/udev/
   ↪ rules.d/
```

A libfreenect2 driver működésének ellenőrzése a teszt program futtatásával. Sikeres konfigurálás esetén a 15. ábrához hasonló eredményt kapjuk:

```
$ ./bin/Protonect
```

15. ábra. Telepítés ellenőrzése Protonect lib segítségével. Balról jobbra haladva: nyers infra, színes és számított mélységi kép együtt, valamint a nyers színes és számított mélységi külön)

A.3. iai_kinect ROS package telepítése

A csomag letöltése, ROS rendszerbe illesztése és buildelése

```
$ cd ~/catkin_ws/src/
$ git clone https://github.com/code-iai/iai_kinect2.git
$ cd iai_kinect2
$ rosdep install -r --from-paths .
$ cd ~/catkin_ws
$ catkin_make -DCMAKE_BUILD_TYPE="Release"
```

A következő három lépést különálló shell-be hajtsuk végre:

ROS OS indítása

```
$ roscore
```

Kinect bridge inditása

```
$ roslaunch kinect2_bridge kinect2_bridge.launch
```

iai_kinect csomag Viewer alkalmazásának indítása

```
$ rosrun kinect2_viewer kinect2_viewer kinect2 sd cloud
```

A.4. VTK, PCL és PyQT telepítése

vtk 5.8 és python vtk wrapper installálása

```
$ sudo apt-get install libvtk5.8
$ sudo apt-get install libvtk5.8-qt4
$ sudo apt-get install python-vtk

$ sudo apt-get install qt4-desinger
$ sudo apt-get install python-qt4-dev
```

PCL és Python PCL wrapper telepítése

```
$ sudo apt install libpcl-1.7-all

cd ~/catkin_ws
git clone https://github.com/strawlab/python-pcl.git
cd python-pcl/
git checkout v0.3.0rc1
sudo python setup.py install --record installed.txt
```