

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar

Szél Péter

LÉZERES TÁVOLSÁGSZKENNER-ALAPÚ LOKALIZÁCIÓS ALGORITMUSOK VIZSGÁLATA

KONZULENS

Kiss Domonkos

BUDAPEST, 2012

Tartalomjegyzék

Összefoglaló **3**

Abstract **4**

Bevezetés **5**

1 A robotplatform bemutatása **7**

2 A rendszerterv **10**

2.1 A hardverterv 12

2.2 Nyákterv 15

3 A lézeres távolságszkenner bemutatása **16**

3.1 Technikai áttekintés (2) 16

3.2 Legfontosabb paraméterek (2) 18

3.3 A paraméterek értékelése 19

4 A szöghisztogram alapú algoritmus megismerése **26**

4.1 Szöghisztogram 26

4.2 A Keresztkorreláció 28

4.3 Az algoritmus 29

4.4 Megvalósítás 33

4.5 Miért nem működik? 34

4.6 Javítási javaslatok 35

4.6.1 A zaj szűrése 35

4.6.2 A szöghisztogram számítási módszer módosítása 38

4.7 A végítélet 38

5 A terepszakasz párosító algoritmusok **39**

5.1 A kiválasztott algoritmus 40

5.2 Adatok szűrése 40

5.3 Jellegzetes tereppontok meghatározása 41

5.4 Jellegzetes pontok párosítása 43

5.5 Pozíció számítása 44

5.6 Mérési eredmények 44

6 Összefoglalás **46**

7 Irodalomjegyzék **47**

Összefoglaló

A mobil robotok egyre nagyobb teret nyernek az iparban és mindennapjainkban egyaránt. Ezek helymeghatározására és navigációjára többféle módszer is napvilágot látott az elmúlt években, évtizedekben. Azonban a hagyományos odometriai módszerek, melyek alapja leggyakrabban az inkrementális adó, sok esetben nem alkalmazhatóak, így szükségessé válik más módszerek vizsgálata. A téma célja egy olyan kutatási célú robotplatform kifejlesztése, amely omnidirekcionális meghajtással rendelkezik, és egy lézeres távolságszkenner segítségével képes információt nyerni a környezetéről. Miután megismerkedtem és használatba vettem a tanszéken található lézeres távolságszkennert, segítségével két lokalizációs algoritmust vizsgáltam meg. Az első Gerhard Weiß, Christopher Wetzler, Ewald von Puttkamer, német kutatók által leírt algoritmus, a másik Kai Lingemann, Andreas Nüchter, Joachim Hertzberg, Hartmut Surmann, szintén német kutatók eredménye. Az algoritmus előbb Matlab segítségével készítettem el, majd implementáltam C nyelven is a robotplatformra, végül valós mérési eredményeket alapul véve értékeltem azokat.

Abstract

The mobile robots are gaining more and more importance in industry and everyday life as well. Several tracking and navigation methods have been published in recent years, decades. But the traditional odometry methods, which are typically based on a rotary encoder, cannot be applied, it becomes necessary to investigate other methods.

The topic is to develop a robotic platform for research, equipped with an omnidirectional drive, a laser range scanner, to be able to gain information about the environment.

First I learned to use the laser rangefinder of the department. With help of the scanner I examined two algorithms, the first one is described by german researchers Gerhard Weiß, Wetzler Christopher, Ewald von Puttkamer, the second one is from Kai Lingemann, Andreas Nüchter, Joachim Hertzberg and Hartmut Surmann. First I implemented the algorithms with Matlab, then implemented in C language for the robot platform, and in the end I evaluated them based on the measured values.

Bevezetés

A mobil robotok egyre nagyobb teret nyernek az iparban és mindennapjainkban egyaránt. Ezek helymeghatározására és navigációjára többféle módszer is napvilágot látott az elmúlt években, évtizedekben. A sikeres navigáció érdekében fontos, hogy a robot valamilyen modellel rendelkezzen a környezetéről, és ebben el tudja magát helyezni.

A projektem célja egy olyan kutatási célú robotplatform kifejlesztése, amely omnidirekcionális meghajtással rendelkezik, és egy lézeres távolságszkenner, illetve egy Microsoft Xbox Kinect szenzor segítségével képes információt nyerni a környezetéről.

A TDK dolgozatomban ennek a nagy projektnek egy részegységét képző problémát vizsgálok meg. készítésének első félévében a lézerszkennerrel ismerkedtem meg, majd a segítségével történő lokalizációt vizsgáltam, konkrétan algoritmust vizsgáltam meg.

Az első fejezetben bemutatom a kiválasztott robotplatformot, részletezem, hogy mi az, amivel már felszerelten érkezik és milyen kiegészítésekre lesz szükség.

A második fejezetben megismertetem az olvasóval a robot rendszertervét, a rendszertervben szereplő egyes modulok szerepét és konkrét megvalósítását. Ezt amiatt tartom fontosnak leírni dolgozatomban, hogy az olvasó megismerkedhessen a dolgozatom második felében leírt algoritmusok megvalósítási keretrendszerével.

A harmadik fejezet a lézeres távolságszkenner működésének leírásával foglalkozik, a konkrét eszköz tulajdonságaival, azok értékelésével, valamint programból való használatával foglalkozom.

A negyedik fejezet a *Gerhard Weiß, Christopher Wetzler, Ewald von Puttkamer,* német kutatók által leírt algoritmus mutatja be részletesen, előbb a szükséges fogalmakat definiálja, majd lépésről lépésre bemutatja az algoritmus futását, és megoldásokat mutat a jelentkező problémákra. Ezután az előbbiekben megismert algoritmus implementációs kérdéseivel foglalkozom, megvizsgálom a hibás működés okait és javaslatokat teszek ezek javítására.

Az ötödik fejezetben előbb általános gondolatokat írok le a jellegzetes tereppont-alapú algoritmusokkal kapcsolatban, majd e család jellemzőit, szükséges lépéseit egy

konkrét algoritmus keresztül mutatom be, hasonlóan az előző fejezethez itt is kitérek az implementációs nehézségekre, azokra nyújtott megoldásaimra.

A hatodik fejezetben egy rövid összefoglaló található, mely egyben tartalmazza a két megismert algoritmus összehasonlítását is, végül értékelem munkám eredményét.

1 A robotplatform bemutatása

Omnidirekcionális meghajtásúnak tekintünk minden olyan, kerekeken mozgó mobil robotot, amely orientációjától függetlenül képes tetszőleges irányú mozgás megvalósítására. A beszédes nevű (*omni* – minden, *direction* - irány) robottípus az egyik legnépszerűbb a holonomikus[1] robotok között, köszönhetően a viszonylag egyszerű és olcsó mechanikai megvalósításnak. Kiváló ellenpélda a hétköznapi közlekedéshez használt ackermann kormányművel ellátott gépjármű, melynek csak párhuzamosan irányítható az első két kereke, ennek megfelelően oldalirányba nem képes mozogni. Ha omnidirekcionális személygépjárműveink lennének, akkor egy kezdő sofőrnek sem okozna nehézséget a parkolás, hiszen egyszerűen oldalra mozogva elfoglalhatná a kínálkozó helyet.

Érdekesség: Az omnidirekcionális meghajtás nem csak a mobil robotok terén érdekes, mint kutatási terület, hanem számos ipari alkalmazása van. Létezik omnidirekcionális kerekekkel ellátott targonca, amelynek nagy előnye a hagyományos társaihoz képest, hogy szűk raktárban sokkal mozgékonyabb, olyan helyekre is be lehet menni vele pakolni, ahová egy ackermann kormányművel ellátott targoncával nem.

Az omnidirekcionális mozgás megvalósításához speciális kerekeket használunk, melyek kerületét szabadon futó görgőkkel látták el. Legegyszerűbb esetben ezek a görgők a kerék síkjára merőlegesen helyezkednek el, ahogyan a következő ábrán is látható:

[1] Egy robotot akkor nevezünk holonomikusnak, ha rendelkezik legalább annyi független beavatkozó szabadsági fokkal, ahány szabadsági fok mentén irányítani kívánjuk. Mivel esetünkben sík terepen mozgó robotról van szó, a kívánt szabadsági fokok száma három (x, y irányú mozgás és az orientáció).



1. ábra - Omni-kerék

A görgőknek köszönhetően a kerekek képesek a tengelyükkel párhuzamosan is elmozdulni. Az ilyen kerekeket omnidirekcionális, vagy csak röviden omni-kerekeknek (*Omni-wheel*) nevezik.

A megoldás legkomolyabb hátránya a kerekek gyakori megcsúszása. Emiatt sajnos a kerekek elfordulásából leszármaztatott helymeghatározási megoldások nagyon hamar pontatlanná válnak. További hátrány, hogy minden görgő súrlódik, így a motorok hatásfoka alacsonyabb a hagyományos kerekek meghajtásához képest, valamint az egyes kerekek sebességének számítása számításigényesebb.

A kiválasztott robotplatform a Nexus Robot[2] cég terméke, amely számos robot alkatrészt gyárt, kerekeket, szenzorokat, mikrokontroller lapokat, és egyéb alkatrészeket, valamint ezek kombinációja is megvásárolható egy csomagban, un. robot kitben. Az általunk megrendelt kit[3] (4WD 100mm Mecanum Wheel Learning Kit 10009) a következőket tartalmazza:

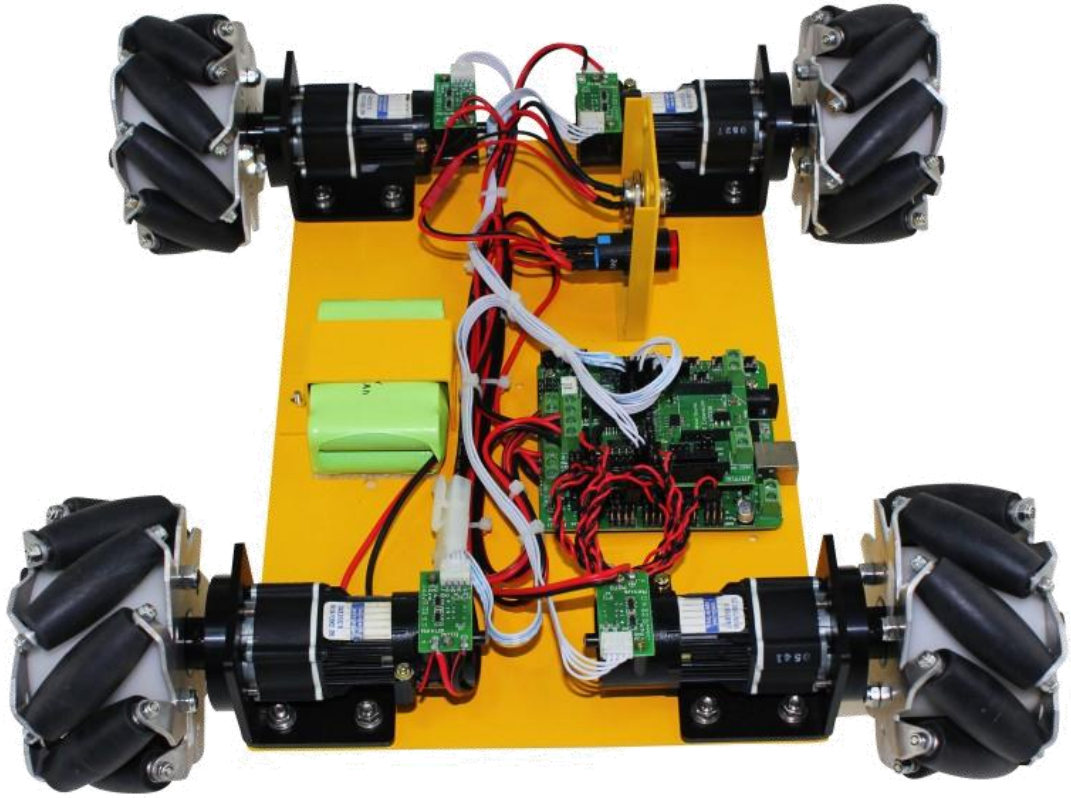
- 4 x 100mm Aluminum Mecanum kerék
- 4 x 12V DC coreless motor
- Arduino 328 vezérlő
- Arduino IO expansion V1.1
- 12V Ni-Mh akkumulátor
- 12V-os töltő

[2] http://nexusrobot.com/

[3] http://nexusrobot.com/product.php?id_product=60

A gyártó leírása szerint a kifejezetten hallgatóknak szánt kit enkóderekkel ellátott motorokkal érkezik, alumínium ötvözet alvázzal.

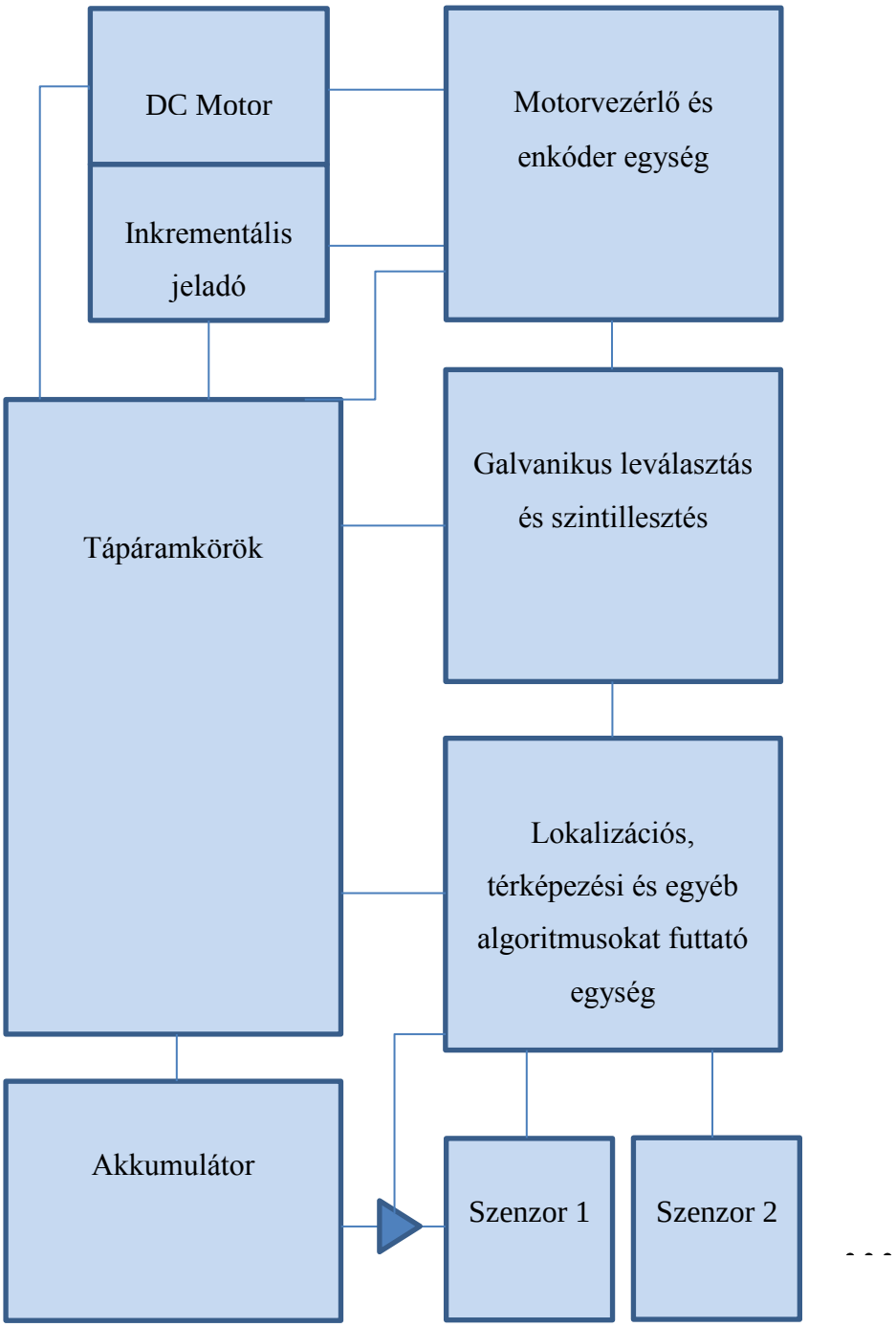
A vezérlő elektronika az Ardiuno platformon alapul, amely egyszerű hardverfelépítésével és programozhatóságával vívta ki magának a megérdemelt népszerűséget. A lapka szíve egy 8 bites, 20 MHz-es AVR mikrokontroller (ATmega328), amely rendelkezik a legalapvetőbb perifériákkal, mint számlálók, USART, SPI és egy 6 csatornás 10 bites AD átalakító. Ezáltal kiválóan alkalmas DC motorok egyszerű szabályozásainak megvalósítására, azonban a következő fejezetekben ismertetett lokalizációs algoritmusok már túl nagy falatot jelentenének számára. Hiányzik továbbá az USB és az Ethernet periféria, pedig a robot különböző részegységeit sokkal célszerűbb ezekkel a modernebb általános kommunikációs protokollokkal összeköttetni. Mindebből látszik, hogy az elektronika kiegészítésre lesz szükség.



2. ábra - A megrendelt robotplatform

2 A rendszerterv

Ebben a fejezetben röviden bemutatom, hogyan áll össze az a rendszer, amelyhez a lézeres távolságszkenner csatlakozik, hogy az olvasó betekintést nyerhessen a projekt teljes képébe. Az alábbi ábrán a robot rendszertervét szemléltetem.



Ahogyan az előző fejezetben is írtam, a robotplatformon 4 motor található, hiszen az omnidirekcionális mozgáshoz négy külön meghajtással rendelkező kerék szükséges. Minden egyes motorhoz tartozik egy inkrementális jeladó is.

A motorok szabályozását külön egység végzi, ennek feladata az inkremetális adók jeleinek feldolgozása, ebből az egyes kerekek sebességeinek számítása. Ennek segítségével megvalósíthatók különféle sebesség visszacsatoló szabályozások. A beavatkozás PWM jel segítségével történik, ennek előállítása szintén az egység feladata.

A sebességszabályozáshoz az alapjelet a lokalizációs, térképezési és navigációs algoritmusok kimenete biztosítja. Ezen algoritmusok jóval bonyolultabbak, így nagyobb számítási kapacitást igényelnek, mint amire az előző fejezetben említett 8 bites mikrokontroller képes. Emiatt feltétlenül szükség van egy másik, jóval nagyobb számítási kapacitással rendelkező egység beszerelésére. Ezzel az egységgel szemben további elvárásaink is vannak, alkalmasnak kell lennie a külvilággal való kapcsolattartásra. Ezalatt egyrészt értem a különböző szenzorokat, pl. a következő fejezetben ismertetett lézeres távolságszkennert, amely etherneten keresztül kommunikál, másrészt a felügyeleti eszközöket, amelyek számára naplózási is hibakeresést megkönnyítő adatokat kell szolgáltatnia. Ez utóbbi tulajdonképpen a fejlesztői PC, amelyhez vezeték nélküli kapcsolaton keresztül kell csatlakoznia, hiszen igen körülményesen kivitelezhető csak egy mobil robot esetében a vezetékes kapcsolat.

A fent leírt elvárásokat sorra véve a választás egy BeagleBoard-Mx kártyára esett, amely egy ARM Cortex A család beli mikrokontrollerrel van ellátva, és a fent leírt össze kritériumnak megfelel. A rajta futó Linux alapú operációs rendszernek köszönhetően kényelmesebb fejlesztői felületet nyújt, mint ha közvetlenül kellene a mikrokontrollert programozni.

Viszont ez a döntés több kihívást is rejt magában. Meg kell oldani a motorvezérlő egység és az algoritmusokat futtató egység közötti kommunikációt. Az első kihívás abban rejlik, hogy míg a BeagleBoard 1,8 V-os jelszinteket használ, melyek magas szinten legalább 1,35 V nagyságúak, addig az Arduino 3,3 V-os jelszintekkel operál, amely legalább 2,9 V-os lesz. Ez alapvetően megoldható egy szintillesztő áramkörrel. Azonban fel kell készülni arra az esetre is, hogy a vezetékeket nem tudjuk úgy vezetni, hogy ne legyenek túl közel a motorokhoz, ezáltal egy 1,8 V-os jelszintet akkora zajterhelés érheti, hogy működésképtelen lesz a kommunikáció. Erre úgy készültem fel, hogy a szintillesztők mellé betettem egy RS 485 transceiver is, amely a differenciál jelátvitelnek

és a csavart érpárnak köszönhetően sokkal nagyobb zajtűréssel rendelkezik. Szerencsére a motorvezérlő egység már gyárilag tartalmazza az RS 485 transceivert.

A következő kihívást a tápellátás kérdése jelenti. Ahogyan az majd a következő fejezetben olvasható, a lézeres távolságszkenner 24 V-ról, a BeagleBoard 5 V-ról, a motorok pedig 12 V-ról működnek. Nem szerencsés megoldás a helykihasználtság szempontjából, ha több akkumulátor is helyet kap, így egy darab, 24 V-ot szolgáltató akkumulátorról kell előállítani mind a három feszültségszintet. Ehhez két tápáramkörre lesz szükség, az egyik 12 V-ot, a másik pedig 5 V-ot fog előállítani.

Tehát a fent leírtak összegzéseként az általam elkészített kiegészítő lappal szemben a következő követelmények vannak:

- Tartalmazzon szintillesztést: 1,8 V – 3,3 V
- Tartalmazzon egy RS 485 transceivert
- 24 V-os akkumulátorfeszültségből elő kell állítania
  - 5 V-ot a BeagleBoard számára (2 A-es terhelhetőség)
  - 12 V-ot a motorok számára (6 A-es terhelhetőség)
- Legyen egy kivezetés rajta a lézerszkenner számára, ezen keresztül biztosítva legyen mind a kézi, mind a szoftverek ki és bekapcsolása a lézerszkennernek
- Csatlakozókat az egyes egységekhez
- néhány alapvető védelmet: túlfeszültség, fordított polaritás ellen

2.1 A hardverterv

A szintillesztéshez a Texas Instruments SN74AVC4T245 típusjelzésű termékét választottam kiváló paraméterei miatt. Az általunk kívánt tartományban 200 Mbps-os sebességre képes, így még a leggyorsabb SPI kommunikációt is probléma nélkül kapcsolja. Figyelnem kellett arra, hogy a BeagleBoard specifikációja szerint a magas jelszint minimuma 1,35 V, azaz egy „hagyományos” 1,8 V – 3,6 V-os illesztő nem biztos, hogy teljesíti ezt az elvárást, tipikusan 1,6 V szokott lenni a magas szint minimum értéke. A kiválasztott chip azonban teljesíti ezt az elvárást, már 1,2 V-os értékre magasat érzékel, így számunkra tökéletes választás. Fontos szempont volt még a választás során, hogy

kedvező árú legyen, ezt a maga nettó 333.77 Ft-os[4] árával teljesíti. (Ez főleg úgy igaz, hogy a TI-tól ingyen rendelhető mintapéldány.) Ezen kívül rendelkezik beépített ESD védelemmel is, ami igen hasznos tekintettel arra, hogy a NYÁK nem lesz dobozolva, így bárki véletlenül hozzáérve szikrakisülést eredményezhet, ami tönkretehetné a chipet.

Az RS485 transceiver kiválasztásakor fontos szempont volt, hogy tartalmazzon galvanikus leválasztást, és bár így drágább az IC, összességében mégis olcsóbb ez a megoldás, mintha külön galvanikus leválasztó chipeket vásároltunk volna. A választásom a Texas Instruments ISO3082 típusú termékére esett, amely megfelel minden elvárásnak, valamint ugyanúgy, mint a szintillesztő, tartalmaz ESD védelmet is.

A tápellátás kérdése már egy kicsivel nagyobb fejtörést okozott, hiszen nem átlagos eset, hogy 24 V-ból úgy kell 12 V-ot csinálni, hogy az hatékony legyen, ugyanakkor akár 6 A áramot is képes legyen elviselni. Értelemszerűen szóba sem kerülhetnek lineáris feszültség átalakítók, csakis kapcsoló üzemű tápegységben lehet gondolkozni. Viszont egy 6 A-es kapcsolóüzemű táp IC-hez nagyon sok külső elem szükséges, sokkal célszerűbb egy kész modult vásárolni, amely önmagában tartalmazza a legtöbb szükséges külső elemet. Egy ilyen modul már viszonylag drágább, viszont ebből is lehet ingyenes mintát kérni a Texas Instrumentstől, így a választásom a PTN78020W típusú modulra esett. Csak két szűrő kondenzátor, valamint egy feszültség beállító ellenállásra van szükség hozzá külső elemként. Ez utóbbi beállítására a modul adatlapja a következő képletet adja:

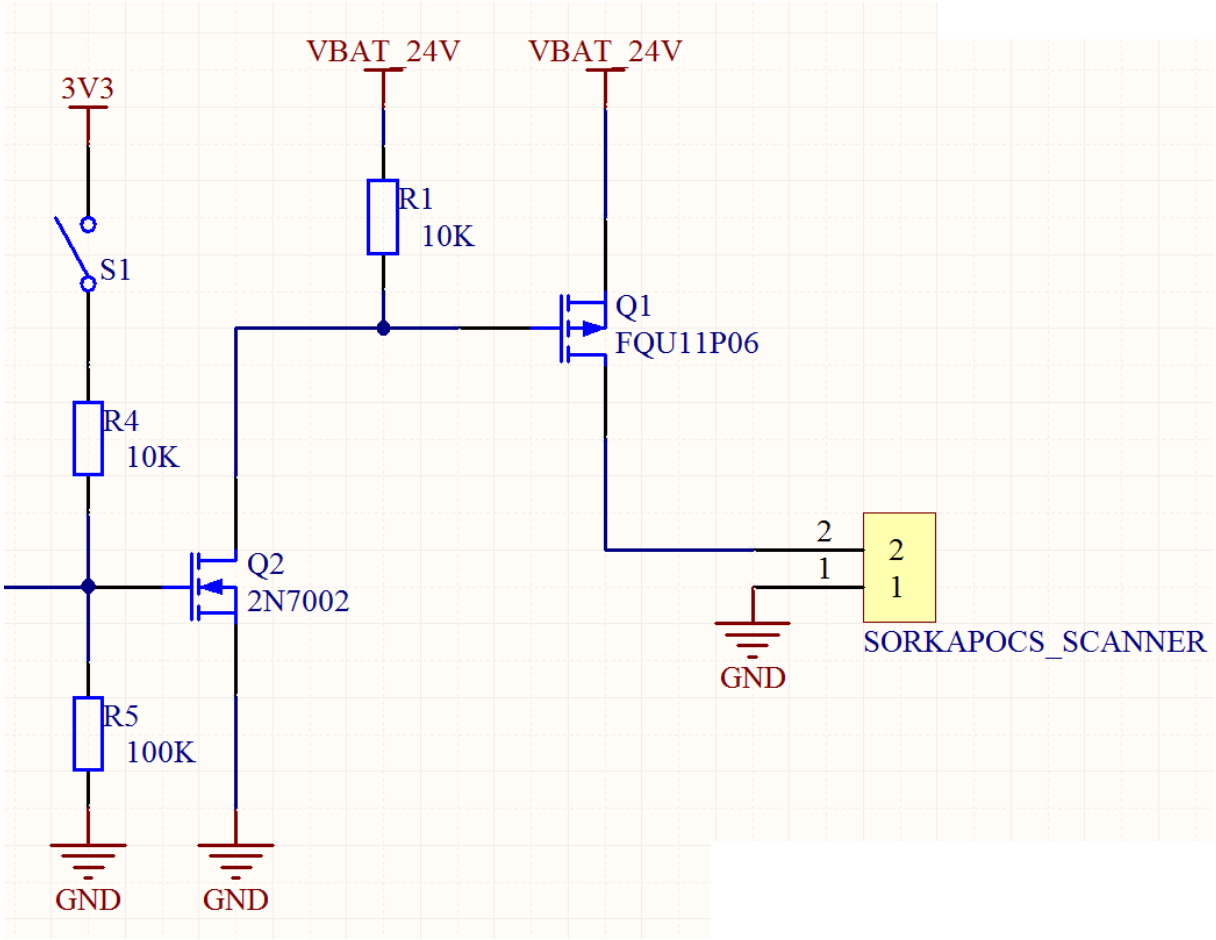
$$R_{SET} = 54.9\ k\Omega \times \frac{1.25\ V}{V_O - V_{min}} - R_P$$

Ennek megfelelően az 5 V előállításához 21 kΩ-os, 12 V előállításához pedig 732 Ω-os ellenállásra lesz szükség.

Emellett beépítettem néhány védelmet, egyrészt egy Schottky diódát a fordított polaritás elleni védelemként, valamint egy kétirányú tranziens elfojtó diódát, a tranziensek ellen.

Érdemes még megemlíteni azt a megoldást, amely a lézerszkenner ki- és bekapcsolását végzi:

[4] http://hu.farnell.com/texas-instruments/sn74avc4t245pw/ic-4bit-bus-transceiver-dual-smd/dp/1053499 (2012. október 25.)



3. ábra - Lézerszkenner ki- és bekapcsoló áramkör

A Q2-es tranzisztor gate-jére érkező jel a BeagleBoard felől jön, de előbb egy szintillesztőn megy keresztül, hiszen egy hagyományos tranzisztort nem lehet kinyitni egy 1,8 V-os feszültséggel.

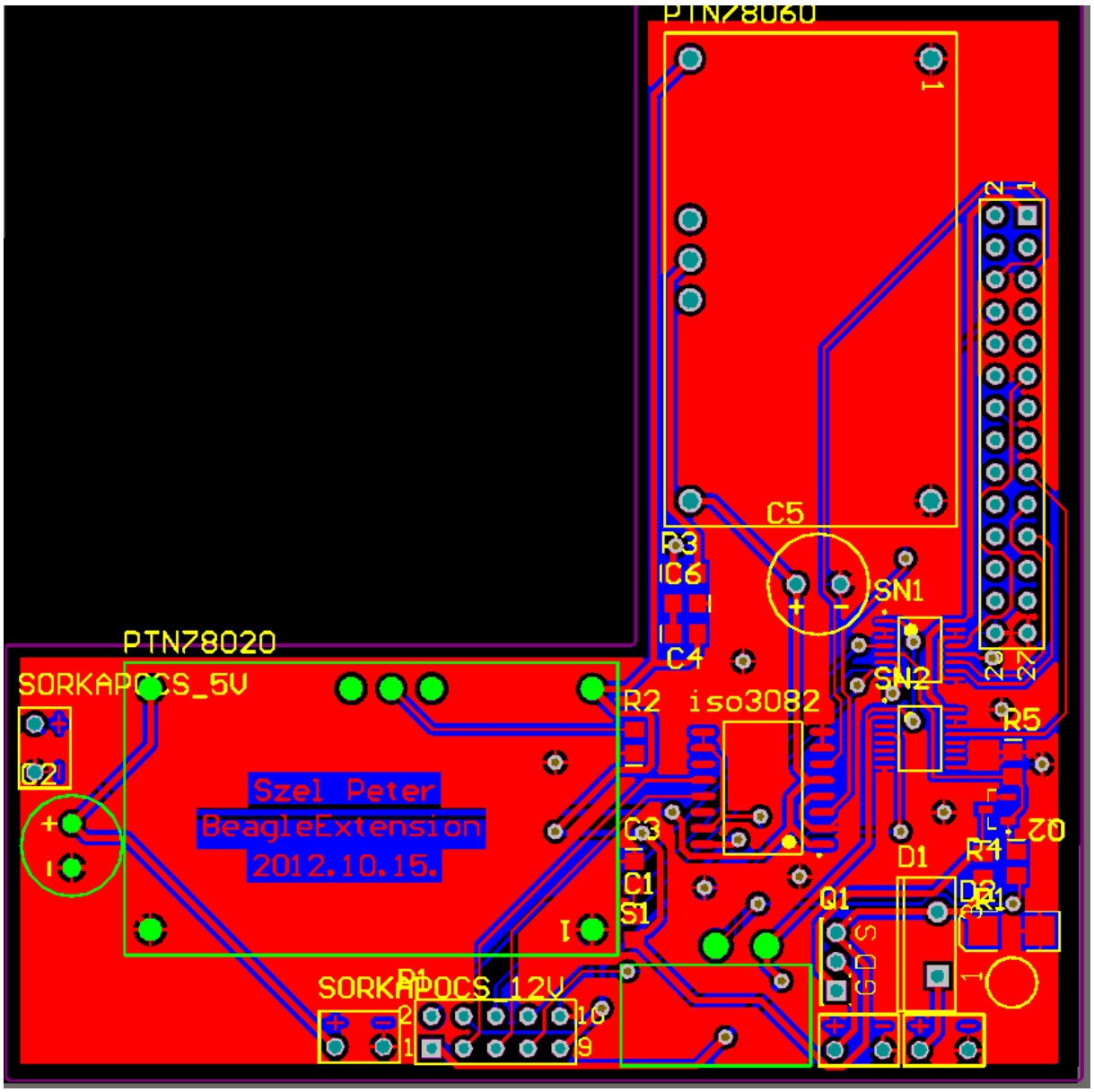
A Q1-es tranzisztor egy p csatornás MOS FET, amely akkor nyit ki, ha a GS feszültsége negatív. Alapból a Q1 gate-jét felhúztam a source-szal azonos szintre, a tranzisztor zárt állapotban van. Ha a Q2 tranzisztor kinyit, úgy a Q1-es gate feszültsége megközelítőleg 0 lesz, így a Q1 tranzisztor kinyit.

A Q2 tranzisztor pedig akkor nyit, ha a GS feszültsége pozitív lesz. Ennek elérésére két lehetőségünk van, vagy az S1 kapcsoló bekapcsolásával, vagy a BeagleBoardtól érkező jellel szoftveresen.

Az így kapott megoldással megvalósul, amit elvártunk, mind kézzel, azaz egy kapcsolóval, mind szoftveres úton be lehet majd kapcsolni a lézerszkennert.

A konkrét tranzisztorok kiválasztásánál arra kell figyelni, hogy a Q1-en keresztül több amperes áram is folyhat, valamint akár 28 V-ot is el kell üzemszerűen tűrnie. Az ilyen tranzisztorokat általában power MOSFET néven találjuk meg a boltokban.

2.2 Nyákterv



4. ábra - Nyákterv

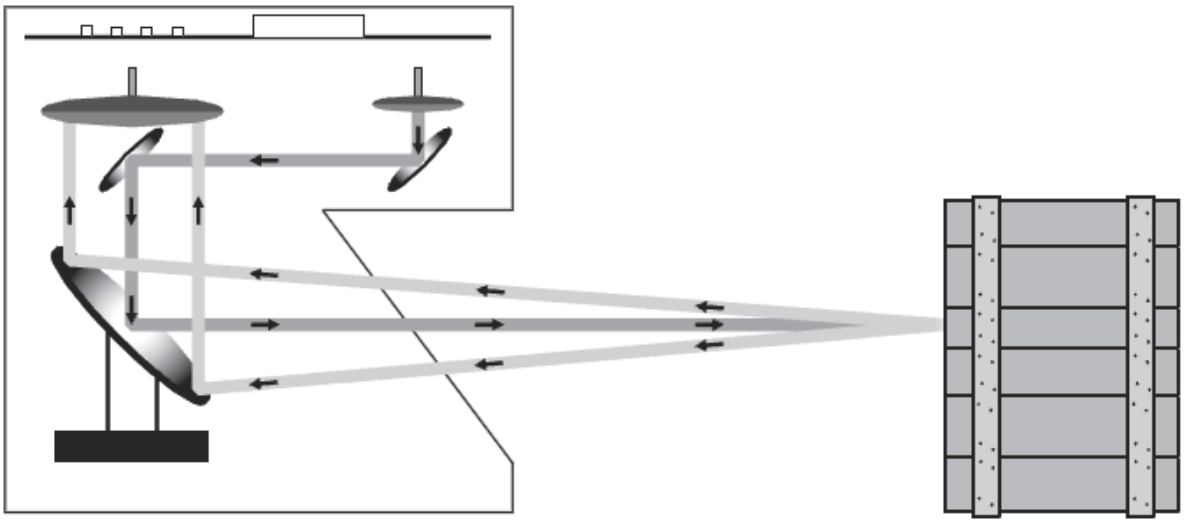
3 A lézeres távolságszkenner bemutatása

A tanszéken található, Leuze rotoScan ROD4-50plus típusú lézeres távolságszkennert használtam az algoritmusok vizsgálatához, valamint ezt a szkennert szereltük fel a robotplatformra. Az eszköz működése, paraméterei, és tulajdonságai alapvetően meghatározzák az algoritmusok hatékonyságát, így ennek a fejezetnek a célja bemutatni és értékelni a lézerszkennerhez tartozó legfontosabb adatokat.

Megjegyzés: Érdekes módon ennek a típusú szenzornak nincs egységesen elfogadott és használt elnevezése az angol nyelvű irodalomban. Egyik legrégebbi és gyakori megnevezése a LIDAR (Light Detection And Ranging), vagy a LADAR (Laser Detection and Ranging). (1) Elterjedt még a „Laser Range-finder Scanner” megnevezés is, vagy ennek egyszerűsítéseként előálló „Laser Scanner”. Ezzel szemben a mi eszközünk adatlapjában az előbb említett kifejezések meg sem jelennek, ott „Area Scanning Distance Sensornak” nevezik. Magyar irodalom híján a lézeres távolságszkenner megnevezést az előbbiek kvázi tükörfordításaként állítottuk elő, de az egyszerűség kedvéért leggyakrabban csak lézerszkennerként fogok a szenzorra hivatkozni.

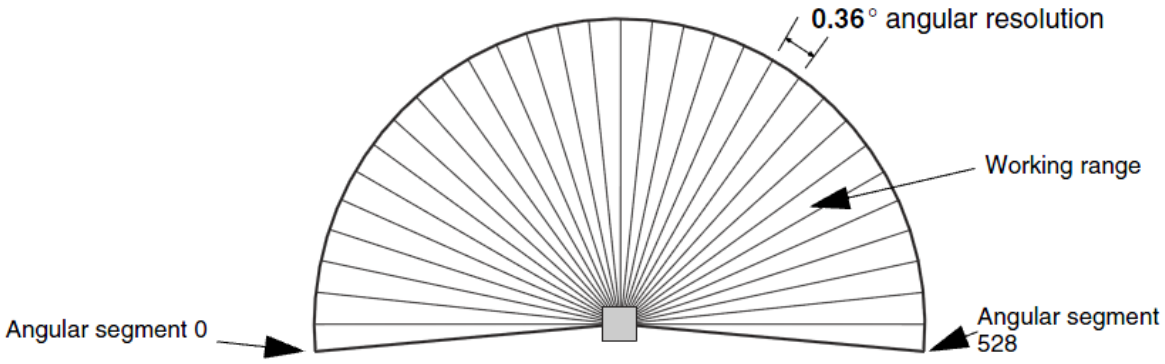
3.1 Technikai áttekintés (2)

Az eszköz félkörív mentén fény impulzusokat bocsájt ki periodikusan egy forgó terelő tűkör segítségével. Ha a fény visszaverődik egy objektumról vagy akadályról, akkor a szenzor érzékeli és analizálja. A visszavert fény terjedési ideje, valamint a terelő aktuális szöge alapján kiszámításra kerül az objektum pontos koordinátája.



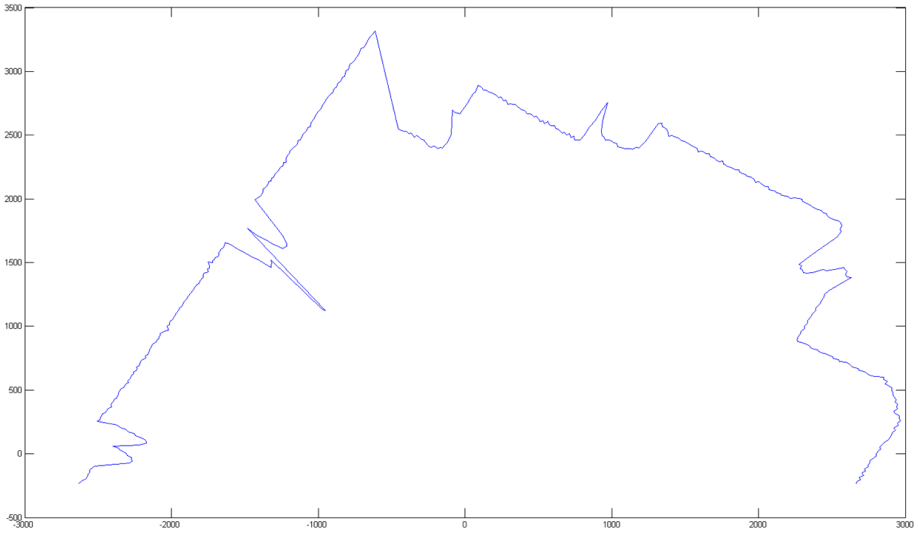
5. ábra - Működési elv (2)

A ROD4plus 190°-os működési tartománya 0,36°-os szegmensekre van osztva, ennek megfelelően 529 ponton kapunk mérési eredményt. A fénynyalábot egy lézer dióda segítségével generálja, majd optikai áttételeken keresztül jut el egy forgó tükörhöz.



6. ábra - Hatáskör és a felbontás (2)

A következő képen látható az eredmény:



7. ábra - A lézeres távolságszkenner kimenete Matlabbal ábrázolva

A felvétel a BME Q épületének QB120-as szobájában készült, legkönnyebben a falak ismerhetőek fel, de a sarokban látható szekrény, valamint a mellette elhelyezett kuka is látható.

3.2 Legfontosabb paraméterek (2)

- működési tartomány: 190^{o}
- W x D x H: 141mm x 168mm x 167mm
- tömeg: 2,3 kg
- áramfelvétel: 1 A (2,5 A-es tápot javasol az adatlap)

Mérési paraméterek:

- Távolságmérés felbontása: 5mm
- Szögmérés felbontása: 0,36^{o}
- Ismételhetőség: ± 15mm
- Hatótávolság: 65 m
- Mérési idő: 20 ms

Számunkra esszenciális be- és kimenetek:

- Tápfeszültség: 24VDC
- kommunikációs interfészek:

- Ethernet: 10/100 MBd, TCP/IP peer-to-peer
- RS 232 115200 baud, 8N1

3.3 A paraméterek értékelése

A legelső paraméter, amely számomra első olvasatra szemet szúrt, az a mérési idő. Ez különösen lényeges paraméter, hiszen ehhez mérten kell majd megválasztani a robothoz kapcsolódó összes időállandót, így a szabályzó algoritmusét is. Ennél rövidebb időállandót értelemszerűen nem lesz érdemes választani, a legszerencsésebb eset az, ha minden egyes mintavételre le tud futni az algoritmus. Emiatt a mérési idő egyben megadja a felső korlátját a lokalizációs algoritmus futási idejének, valamint végső soron ez fogja megszabni a robot mozgásának lehetséges végsebességét.

Első hallásra az ember a 20 ms-os mérési időt lassúnak érzi a hagyományos odomeriához képest, ott jóval gyakrabban állnak rendelkezésre mérési adatok (ha szeretnénk, akkor akár impulzusonként), ez mindenképp egy komoly korlátot jelent. Viszont ha a másik oldalról közelítem meg a kérdést, a robot omnidirekcionális meghajtása miatt biztosan nem fog tudni gyorsabban haladni, mint 1,5 m/s [5], azaz 20 ms alatt mindösszesen 30 mm-t fog tudni megtenni a legjobb esetben is, ez pedig teljesíthető elvárás egy lézerszkenneres lokalizációs algoritmussal szemben.

Tehát az aktuális robotplatformhoz elég a lézerszkenner mérési ideje, azonban nem árt megjegyezni későbbi kutatásokhoz, hogy ha például egy 40 km/h-ás (11,1 m/s) végsebességre képes RobonAUT[6]-os autóra szeretnénk megvalósítani lokalizációt a ROD4plus segítségével, már valószínűleg komoly nehézségekbe fogunk ütközni.

A másik kulcsparaméter a távolságmérés felbontása. Ismét térjünk vissza a hagyományos odometriához viszonyításképpen. A mi robotplatformunkon a motorokra igen egyszerű inkrementális adót szereltek, mindössze 12 osztást tartalmaznak körülfordulásonként (12 CPR). A kerék átmérője 100 mm, azaz az így nyerhető felbontásunk $(100 * \pi)/12 \approx 26$ mm. Pontos lokalizációra egyszerűen alkalmatlan a gyárilag szerelt inkrementális jeladó. Nem is erre találták ki, hanem a kerék sebességének becslésére, arra pedig tökéletesen alkalmas. Ehhez képest a lézerszkenner 5 mm-es felbontása nagyon jónak tekinthető. Ne feledjük azonban, hogy kereskedelemben, a lézerszkennerhez képest elenyésző áron kaphatóak az előbb leírtnál lényegesen

[5] Valójában terhelt állapotban ennek a felével sem képes haladni.

[6] http://robonaut.hu/

pontosabb, akár 4096 CPR-es jeladók, mellyel számolva a felbontás 0,077 mm-re csökkenthető. (Továbbá még nem is számoltunk azzal a lehetőséggel, hogy nem egyszeres, hanem négyszeres kiértékelést használunk.) Világos tehát, hogy a lézeres szenzorunkkal megvalósított lokalizációval csak lényeges alacsonyabb pontosság érhető el, mint hagyományos odometria alkalmazásával.

Érdekes adat még az ismételhetőség, amelyre ± 15mm van megadva. Bár ennél nincs pontosabban definiálva, én ez alapján arra számítok, hogy a jelet egy olyan Gauss eloszlással közelíthető zaj terheli, melynek középértéke nulla, valamint 15mm felett, illetve -15mm alatt elhanyagolható az értéke. A zaj paramétereinek empirikus becslésével külön fejezetben foglalkozom.

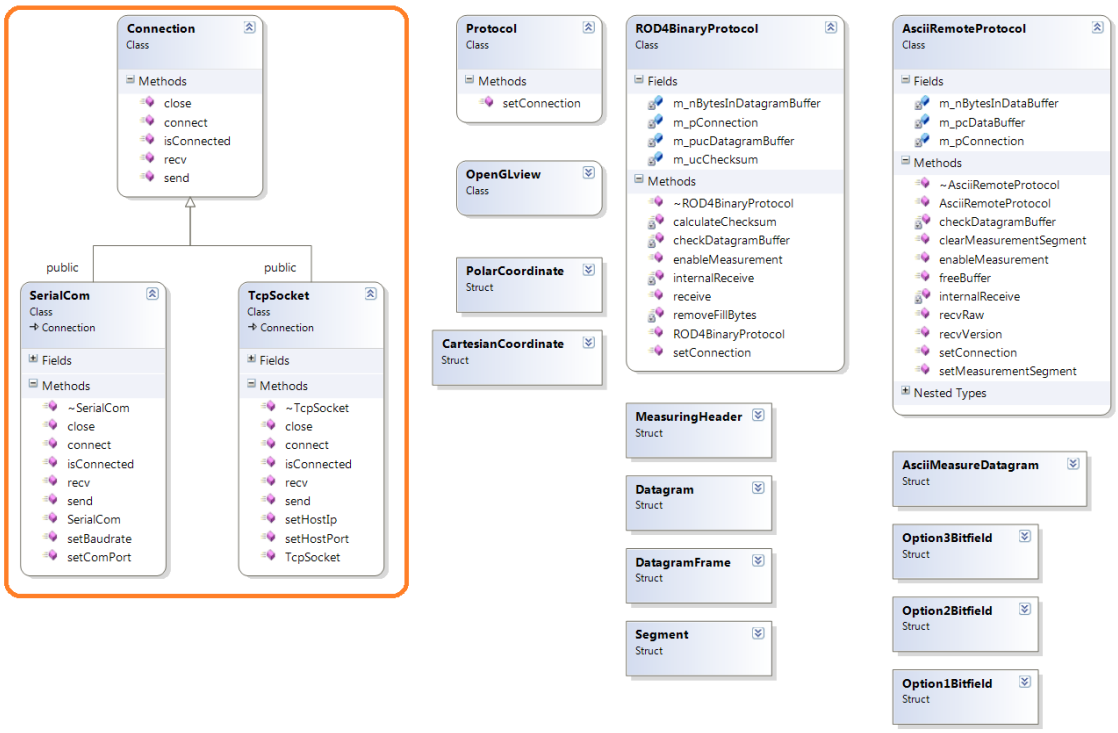
A 24 V-os tápfeszültség nem meglepő, hiszen ez a feszültségszint ipari alkalmazásokban igen elterjedt. Számomra annyiban fog nehézséget okozni, hogy a robotplatformon eredetileg egy 12 V-os akkumulátor van, a motorok is ezen a feszültségen üzemelnek, szükség lesz egy másik, 24 V-os akkumulátorra. Emellett természetesen meg kell oldani minden egység tápellátását.

A kommunikációs interfészek közül nyilván az modernebb, hibatűrőbb és gyorsabb ethernetet fogom használni. Az egyes mérési adatok egy-egy UDP csomagban jutnak el a klienshez, ennek és a peer-to-peer kialakításnak köszönhetően a kommunikáció kvázi real time lesz.



8. ábra - Lézerszkenner

A gyártó biztosít a használathoz egy C++ könyvtárat, viszont ez csak Microsoft Windows alatt működik, és a tervek szerint a roboton egy beágyazott Linux operációs rendszer fog futni. Emiatt ki kellett bővítenem a könyvtárat úgy, hogy Unix és Windows rendszerre egyaránt fordítható legyen. Ehhez először meg kellett ismernem a könyvtár felépítését:



9. ábra - Osztálykönyvtár felépítése

Szerencsére a könyvtár készítői gondoltak azokra, akik más rendszerhez szeretnék illeszteni a könyvtárat, külön választották a platformfüggő és a platform független részeket.

A platformfüggő részeket narancssárga színnel jelölöm az ábrámon, ezek felelnek a kommunikációért. A soros port (*SerialCom*) és a TCP/IP kapcsolat (*TcpSocket*) megvalósításául szolgáló osztályok közös absztrakt ősosztályból (*Connection*) származnak. A kód többi részében csak ezt az ősosztályt használják, így az teljesen független a kommunikáció típusától.

A lézerszkenner két féle protokollon keresztül képes adatot küldeni a számunkra, az egyik bináris, a másik sima szöveg alapú. Az előbbi tömörebb, így gyorsabb, az utóbbi több lehetőséget kínál és rugalmasabb (pl. megválasztható, hogy milyen koordinátarendszerben küldje az adatokat, vagy lehetőség van csak egy-egy térrészlet lekérdezésére). A két protokoll használatához két osztály áll rendelkezésre, valamint számos adatstruktúra.

Kapunk még a könyvtárral a grafikus megjelenítést elősegítő OpenGL segédosztályt. Erre a roboton valószínűleg nem lesz szükség, de később még hasznunkra válhat.

Mivel a soros kommunikációt úgy sem fogjuk használni, így csak a TcpSocket osztályt kell portolnom unixos környezetbe. Ezzel szerencsém van, hiszen mind a Windows, mind a Unix a Berkeley Socket API-t használja a TCP kapcsolat kezeléséhez, így a legtöbb függvény neve és paraméterlistája megegyezik. Ahol nem, ott preprocesszor direktívák segítségével kell különválasztanom a különböző platformokra szánt kódot, például így:

3-1. kódrészlet - Rendszernek megfelelő inclode-ok használata

```
#ifdef WIN32
#include <windows.h>
#endif
#ifdef unix
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <fcntl.h>
#endif
```

Így ha a *WIN32* szimbólum definiálva van (ezt definiálja a Visual Studio), akkor a *windows.h* kerül beágyazásra, ha a *unix* szimbólum (ezt definiálja minden unixra fordító alkalmazás, pl. a GCC is), akkor az ahhoz szükséges könyvtárakat.

Néhány különbség feloldható néhány egyszerű *define* használatával:

3-2. kódrészlet - A unix rendszereken hiányzó definíciók

```
#ifdef unix
#define INVALID_SOCKET -1
#define SOCKET_ERROR   -1
#define closesocket(s) ::close(s);
typedef int SOCKET;
typedef sockaddr_in SOCKADDR_IN;
#endif
```

Látható, hogy a Windows API close helyett closesocket elnevezést használja, viszont a két függvény egyébként ugyan, így csak egy névdefinícióra van szükség. A Windows továbbá definiál pár típust, ami a unixos világban nem ismert, viszont ezek is könnyen pótolhatóak.

A komolyabb problémák ott kezdődnek, amikor két rendszerben teljesen eltér a megvalósítás. A legelső ilyen eset, hogy a Windows megköveteli, hogy a DLL először inicializálva legyen a megfelelő verzióval, erre unixban nincs szükség, tehát a konstruktort ki kell egészíteni:

3-3. kódrészlet - Windows DLL inicializációja

```
#ifdef WIN32
    //initialize ethernet (necessary only first time)
    if(!s_bWsaInitialized)
    {
        //we want version 2.2
        WORD wVersionRequested = MAKEWORD(2, 2);
        WSADATA wsaData;
        int iErr = WSAStartup(wVersionRequested, &wsaData);
        if(iErr)
        {
            MessageBox(NULL, "Could not find usable WinSock DLL", "TcpSocket",
MB_OK);
        }
        else
        {
            //is version 2.2 supported?
            if( (2 != LOBYTE(wsaData.wVersion)) ||
                    (2 != HIBYTE(wsaData.wVersion)) )
            {
                MessageBox(NULL, "WinSock DLL does not support Version 2.2",
"TcpSocket", MB_OK);
                WSACleanup( );
            }
            else
            {
                s_bWsaInitialized = true;
            }
        }
    }
#endif
```

A másik eset, amely nagyon különbözik, az az ahogyan egy IO leírót nem blokkoló módba lehet állítani, itt teljesen különbözően használandó függvények vannak:

3-4. kódrészlet - Nem blokkoló módba állítás

```
#ifdef WIN32
    u_long uArg = 1;
    iErr = ioctlsocket(m_socket, FIONBIO, &uArg);
#endif

#ifdef unix
    int flags = fcntl(m_socket, F_GETFL, 0);              // Get socket flags
    iErr = fcntl(m_socket, F_SETFL, flags | O_NONBLOCK);   // Add non-blocking
flag
#endif
```

Az utolsó különbség, amely figyelmet érdemel, a különböző hibatípusok lekérdezése:

3-5. kódrészlet - Hibakód lekérdezése

```
#ifdef WIN32
        if(WSAEWOULDBLOCK != WSAGetLastError())
#endif
#ifdef unix
```

```
        if(errno != EINPROGRESS)
#endif
        {
            perror("Could not connect to host");
            close();
            return;
        }
```

Ezzel elértem, hogy a könyvtár használható bármilyen Unixos (Linux vagy Mac OS X) és Windowsos rendszerekben is.

Az így kapott osztálykönyvtár fölé készítettem még egy réteget, ami elrejti az összes könyvtárat, egyszerű és egységes felületet nyújtva. Ezzel az volt a célom, hogy ha később majd más szeretné használni a lézerszkennert, sokkal egyszerűbb dolga legyen, ne kelljen az alsóbb rétegekkel foglalkoznia, hanem koncentrálhasson az algoritmus megvalósítására.

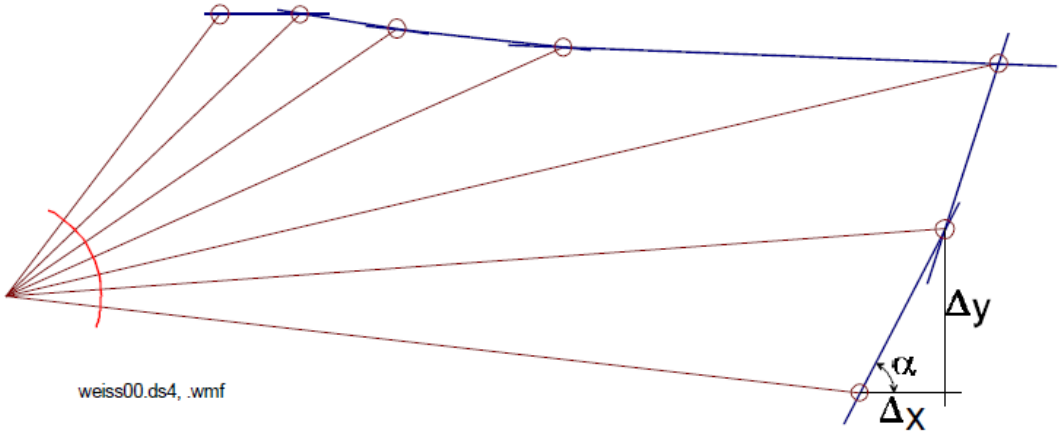
4 A szöghisztogram alapú algoritmus megismerése

A TDK készítésének első félévében a lézerszkenner segítségével történő lokalizációt vizsgáltam, konkrétan *Gerhard Weiß, Christopher Wetzler, Ewald von Puttkamer,* német kutatók által leírt algoritmust vizsgáltam meg. Az algoritmus megértéséhez előbb tisztázok néhány kulcs fogalmat.

4.1 Szöghisztogram

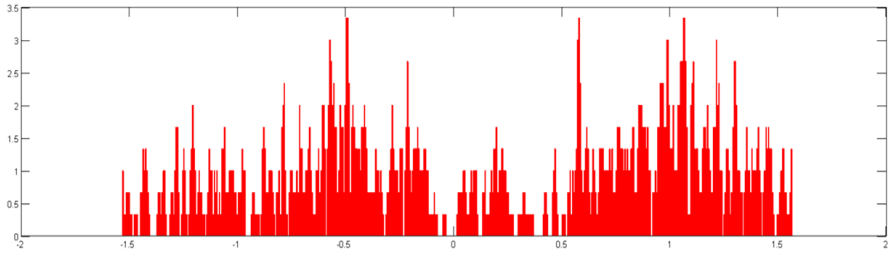
A hisztogram a statisztika egyik alapfogalma, az eloszlásfüggvény diszkrét közelítése. Az egész értéktartományt véges sok részre bontjuk, egy-egy részt rekesznek nevezünk (angolul „bin”). Minden egyes mintát az értékének megfelelő rekeszbe helyezünk, majd ábrázoljuk a rekeszek számosságát sávdiagrammal. A kérdés csak az, hogy mely adatokból készítünk hisztogramot.

Hinkel és Knieriemen 1988-ban megjelent művükben mutatják be a szöghisztogramot. Leírásuk szerint, miután elkészült a lézerszkennelés, kapunk egy térképet (7. ábra) amelyen a legközelebbi tárgyak látszanak. A térkép egyes szomszédos pontjai között kiszámoljuk a relatív szögeket, ahogyan a 10. ábra mutatja.



10. ábra - Szögek számítása a szöghisztogramhoz (3)

Az így kapott szögek hisztogramja, a szöghisztogram, a 11. ábrán látható.



11. ábra – szöghisztogram

Megjegyzés: A fenti ábra, illetve az ezt követő szemléltető ábrák mind saját valós mérési eredményeimet mutatják.

A 11. ábrán a 7. ábra adatain alapuló szöghisztogram látható. Egyértelműen kivehetőek rajta a falak fő irányai a hisztogram maximumainál. Az is látszik azonban, hogy nagyon zajos, ez a szomszédos pontok távolságmérésének hibájából fakad, szükségünk van szűrésre. Ezzel a módszerrel megállapíthatjuk, hogy a robot hivatkozási rendszeréhez képest milyen szögben állnak a falak.

Tovább javíthatjuk a módszert, ha a szögek modulo π értékével számolunk, ekkor ugyanis (feltételezve, hogy a falak ortogonálisak) az egymással szemben elhelyezkedő, vagyis az egymással párhuzamos falak ugyan azon a pontokon jelennek meg a hisztogramon. Ennek köszönhetően csak két nagy csúcsot kapunk. A szerzők rámutatnak, hogy a megoldás igen robosztus, hiszen a falakkal egyvonalban lévő tárgyak, mint például a szekrények, ajtók, ablakok sem zavarják meg a hisztogramot.

Weiß et al. további kísérleteket végeztek a szöghisztogrammal. Munkájuk rámutat, hogy a szöghisztogram nagyjából invariáns a szkenneradatok forgatására és eltolására. Ha csak forgatjuk a képet, az a szöghisztogramban csak fázistolást eredményez. Ha mozgatás is történik, akkor a szögek eloszlása is változik, ugyanakkor továbbra is reprezentálja az fő irányokat. Vagyis ahol a korábbi hisztogramon lokális maximum volt, ott továbbra is maximum hely várható, köszönhetően annak, hogy két szkennelés között a robot elmozdulása meglehetősen kicsiny volt. Ezt a tulajdonságot pedig felhasználhatjuk az orientáció számításánál.

4.2 A Keresztkorreláció

A matematikában a korreláció egyfajta hasonlóságot, lineáris kapcsolatot jelent. Minél magasabb a korreláció értéke, annál inkább hasonlít egymásra a két adatsor („a mérési eredmények korrelálnak”). A korreláció számítására számos módszert dolgoztak ki, az adatsorok jellegétől és a számítás céljától függően. Az egyik ilyen számítása módszer a keresztkorreláció, amely definícióját az alábbi képlet adja meg.

$$c(y) = \lim_{X\to\infty} \frac{1}{2X} \int_{-X}^{X} f(x)g(x+y)dx$$

4-1. egyenlet - A folytonos idejű keresztkorrelációs függvény

A keresztkorrelációs függvény paramétere y, a fázistolást adja meg a két sztochasztikus függvény között. Ha a keresztkorrelációs függvény s helyen veszi fel a maximumát, akkor *g(x)*-et s-el eltolva hasonlít leginkább *f(x)*–re.

Véges időben, és véges N darab mintára az alább látható módon módosul a képlet.

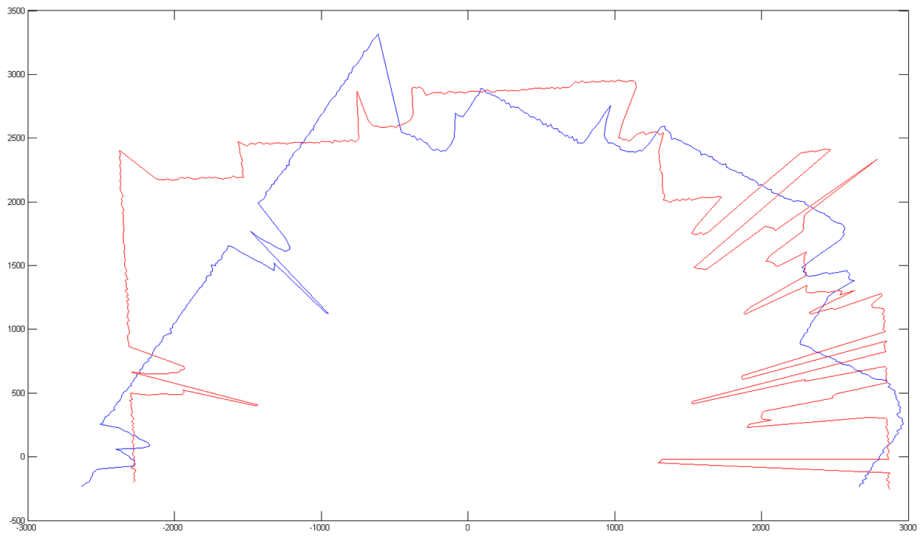
$$c[i] = \sum_{k=1}^{N} f[i]\, g[i+k]$$

4-2. egyenlet - A diszkrét idejű keresztkorrelációs függvény

Rögtön felmerül a kérdés, hogy ugyan csak N darab mintánk van, a képlet alapján mégis N-nél magasabb indexű elemét használjuk a g sorozatnak. Emiatt kibővítjük a fenti képlet értelmezését úgy, hogy *g[i]* alatt, mindig *g[i mod N]*-et értünk. Ebben az esetben is ugyan az igaz, a c[i] véges értéket vehet föl, ezek közül a legnagyobb lesz az az index, amellyel az egyik sorozatot eltolva a leginkább hasonló lesz a két mintasorozat.

4.3 Az algoritmus

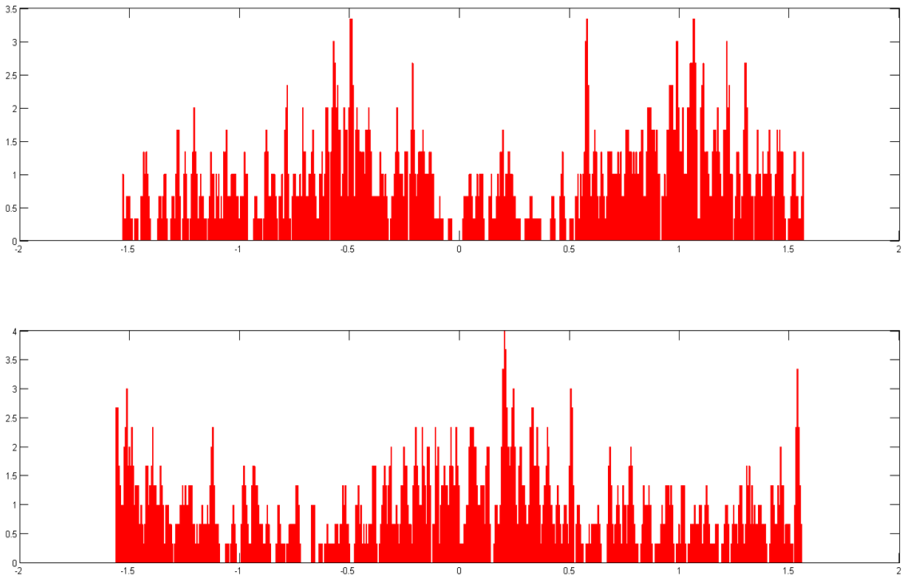
A kulcsfogalmak tisztázása után rátérek az algoritmusra. A kiindulási helyzet, hogy van két mérési eredményem, a második mérés előtt a robot elfordult, az eredményt az ábra szemlélteti.



12. ábra - Az eredeti és elforgatás utáni térkép

Ezen két adatsorból kell kiszámolnom az elfordulást és az elmozdulás mértékét.

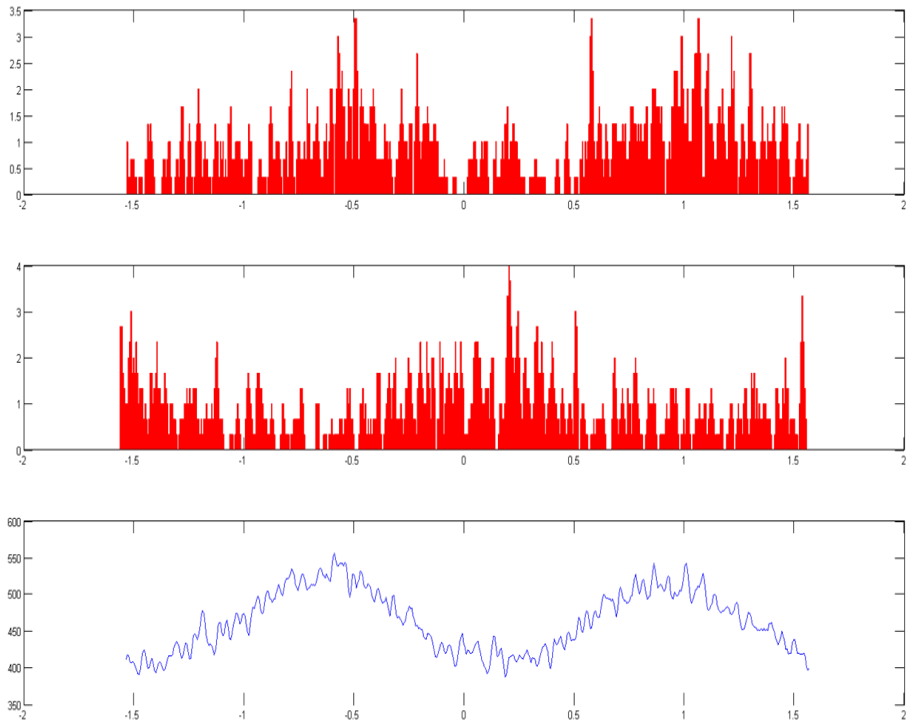
Első lépésként mind a két adatsorra elkészítem a szöghisztogramot (a π–vel vett modulos változatot), végrehajtok rajta egy egyszerű szűrést. Az így kapott két szöghisztogramot az alábbi ábra szemléltet.



13. ábra - Az eredeti és az elforgatott állapot szöghisztogramjai

Jól látszik az, amit a kutatók is jeleztek, az elforgatás hatására fázistolást szenvedett el a szöghisztogram.

Ezek után kiszámítom a két szöghisztogram keresztkorrelációs függvényét, az eredményt az alábbi ábra szemlélteti.



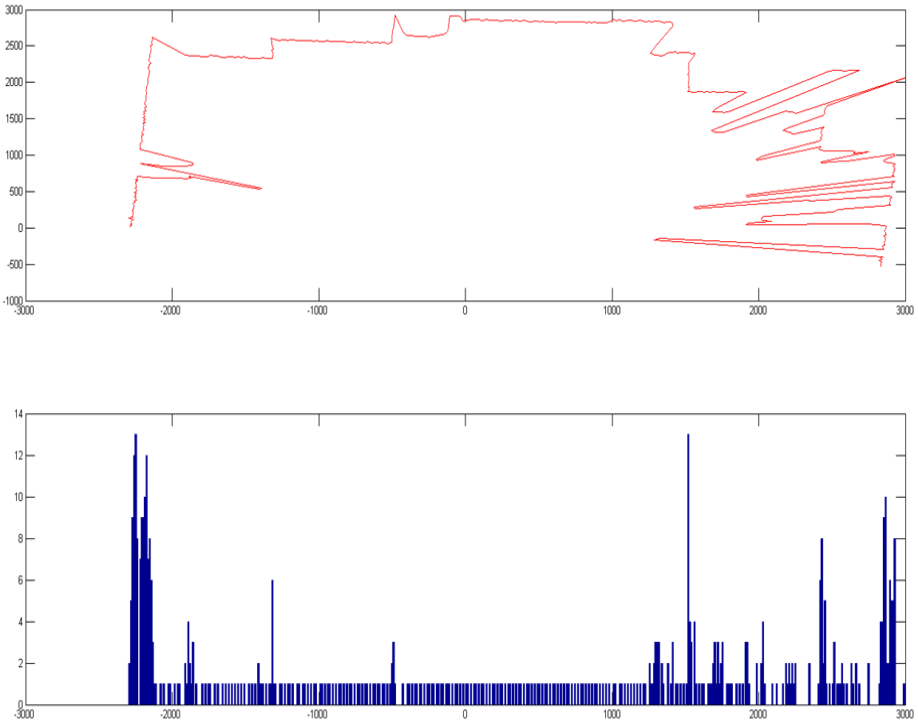
14. ábra - A keresztkorrelációs függvény

A kapott függvény maximum helye adja meg azt a fázistolást, amelyet az első hisztogramon alkalmazva közelítőleg a másikat kapjuk. Így ez lesz az az érték, amellyel elfordult a robot.

Érdemes megjegyezni, hogy két maximuma is lesz a keresztkorrelációs függvénynek, ez abból adódik, hogy a két fal nagyjából azonos szinten van reprezentálva a hisztogramban, így akkor is magasabb értékű korrelációt kapunk, ha az egyik fal által okozott csúcs a másik fal csúcsával kerül átfedésbe. Ebből is következik, hogy a keresztkorrelációs függvény két maximumhelye épp 90^{o}-ra van egymástól (a két fal által bezárt szög). Mivel a robot korlátos sebességgel fog haladni, így a két maximum közül mindig az abszolút értékben kisebbet fogjuk választani.

A fent leírt módszer nagy előnye, hogy igen jó a zajérzékenysége, a fent látható viszonylag zajos hisztogramok alapján is jó értéket adott az orientációra.

Miután az orientációt meghatároztam, a pozíción a sor. A már fent leírt módszer alkalmazhatónak tűnik az x és y koordinátákra is, tehát készítsük el külön-külön a hisztogramokat. Az alábbi ábra az X hisztogramot mutatja.



15. ábra - x hisztogram szemléltetése

Jól látszik, hogy a tengelyre közel merőleges falak kiválóan megtalálhatóak a hisztogramos módszerrel (a kép baloldalán egyértelmű a csúcs), ellenben más orientációjú falak esetén már nem működik a módszer. Emiatt hisztogram készítése előtt minden esetben el kell forgatnunk a térképet úgy, hogy a falak merőlegesek legyenek a tengelyekre. De honnan tudjuk meg, hogy mennyivel kell elforgatni a képet? Pont erre találták ki a szöghisztogramot! A szöghisztogram maximumhelyei megadják a falak orientációját, ezek közül mindig a kisebbet kell választani, hiszen ha 45^{o}-nál nagyobbat forgatunk a képen, akkor felcserérődnek az x és y tengelyek.

Miután megkaptuk az XY hisztogramokat, ugyanúgy, mint a szögek esetén, a keresztkorreláció számítása következik. Viszont van még egy probléma, mégpedig az, hogy az X és Y értékek nem cirkulárisak, azaz nem tekinthetjük úgy, mintha az utolsó érték után az első következne, tehát nem működne helyesen a keresztkorrelációs módszer. Erre a problémára azt a javaslatot adják a szerzők, hogy hajtogassuk be véges méretűre az egész X és Y tengelyt, a következőképpen:

$$f(x) = x \bmod size$$

$$f(y) = y \bmod size$$

A „size” konstanst elég nagyra kell választani ahhoz, hogy a fázis tolás az X és Y hisztogramok között még egyértelmű legyen.

Az így kiszámított keresztkorrelációs függvényeknek a szögekhez hasonlóan itt is több maximuma lesz a konstanssal vett modulonak köszönhetően. Emiatt végeznünk kell egy hihetőségi vizsgálatot, azt a pozíciót hisszük el, amelyik közelebb van a korábban mért pozícióhoz. A kutatók azt javasolják, hogy egy másik szenzor eredményei alapján érdemes egy durva becslést adni az elfordulásra és az elmozdulásra, és ez alapján választani a hihetőbb eredményt.

4.4 Megvalósítás

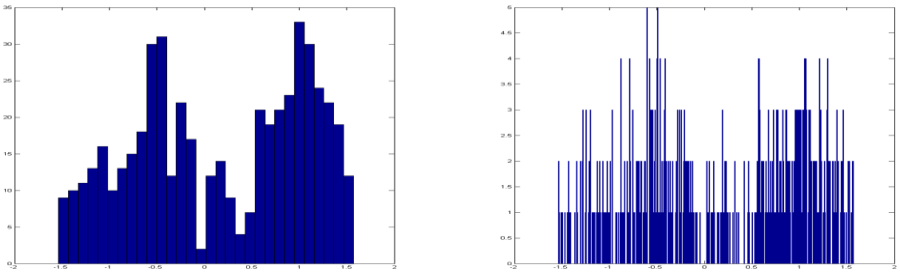
Az algoritmust előbb Matlabbal[7] készítettem el, hiszen a program rendelkezésre bocsájt számos matematikai függvényt, mint a hisztogram számítása, keresztkorreláció számítása, vagy például a maximumkeresés. Emellett hasznos, hogy a kapott eredményeket egyből könnyedén kirajzoltathatom, így könnyebb megkeresni, hogy mi volt a probléma. A fenti ábrákat is mind Matlabbal készítettem.

Miután elkészült az algoritmus, C nyelven is implementáltam, hiszen a robotra ezen a nyelven készítjük a programokat. Az így elkészült kód értelemszerűen jóval hosszabb lett, hiszen számos Matlab beépített függvényt is implementálnom kellet.

Az elkészült programok megtalálhatóak a Mellékletben, itt csak néhány implementációs kérdést említek meg, amely nem szerepelt az algoritmus leíróinak művében.

Az egyik legfontosabb kérdés a futási idő, hiszen ha túl ritkán fut le az algoritmus, akkor nem működhetnek a szabályozások megfelelően. Márpedig az egyből látszik, hogy viszonylag számításigényes algoritmusról van szó. Futási idő leginkább egy olyan paramétertől függ, amiről eddig nem is esett szó (és a szerzők írásában sincs), ez pedig az, hogy hány részre osztjuk az érték készletet a hisztogram készítése során, azaz mekkora a hisztogram felbontása.

[7] MathWorks Matlab: http://www.mathworks.com/products/matlab/



16. ábra - A felbontás szemléltetése

A 16. ábra jól szemlélteti, hogy mekkora jelentősége van a felbontásnak. Míg a baloldali hisztogram esetében mindössze 30 részre oszlik a tartomány, addig a jobboldalinál 500-ra. Szembetűnő a különbség, az első esetben egyértelműen sokkal pontatlanabb lesz a végeredmény. Viszont nem növelhetjük a felbontást minden határon túl, hiszen egy idő után eljutunk, oda, hogy miden osztásba legfeljebb egyetlen minta kerül, ami nem használható semmire. Ennél azonban komolyabb felső korlát a már említett futási idő, amely négyzetesen függ a felbontásról (n), hiszen a keresztkorreláció számításánál n-szer kell eltolni a hisztogramot, és minden eltolás után n darab szorzást kell végezni. Ezt a gondolatot jól visszaadják a mérési eredményeim is, az első esetben 500-as felbontásnál 8-9 ms alatt futott le, 250-es felbontásnál 2-3 ms alatt.

A másik fontos paraméter, amely megválasztást igényel, az az XY hisztogram esetében alkalmazott „size” konstans. A felbontás és a „size” változó erősen összefügg. A „size”-nál alsó korlátot szab az, hogy maximális sebességnél mennyit tud haladni a robot két mérés között, e fölé kéne belőn. Ha van egy pontossági korlát (mennyire durva felbontást engedünk meg), akkor a „size”-ból és ebből kijön, hogy hány részre bonthatod fel „size”-ot, hogy még elég pontosak legyünk.

4.5 Miért nem működik?

Sajnos az implementáció során meg kellett tapasztalnom, hogy a fent bemutatott algoritmus nagyon komoly problémákkal küzd. A valós körülmények között futó mérések során arra az eredményre jutottam, hogy a szög számítása ugyan legfeljebb ± 3^{o}-os hibával, de többnyire pontos eredményt ad, viszont az elmozdulás számítása csak időnként pontos, többnyire meg sem közelíti a valós értékeket.

Hosszabb vizsgálódás során jöttem rá, hogy a problémát a zaj okozza. Ez nem volt egyértelmű, mivel a mérési eredményünk zajos ugyan (már fent ismertettem a

specifikációban megaadott ± 15 mm-es intervallumot), de ez messze nem okoz akkora hibát a távolság mérésben, amekkora tapasztalható a végeredményben. A problémát az okozza, hogy a szög számítási módjából adódóan távolságmérés kisebb hibái a szöghisztogram számítása során számottevően nagy hibát eredményezhetnek. Ez megfigyelhető a fenti ábrákon is, amíg a 7. ábrán ebben a nagyításban nem látszik a zaj, addig 11. ábra már igen zajos, felismerhető a jellegzetes haranggörbe alak.

A keresztkorreláció robosztusságából adódóan a robot elfordulásának szöge a nagy zaj ellenére viszonylag pontosan adódik, ezzel nincsen probléma. Azon a ponton téved nagyot az algoritmus, amikor szöghisztogram maximumát keresve meg szeretnénk állapítani, hogy mennyivel kell elforgatni a képet ahhoz, hogy párhuzamosan álljanak a falak a tengelyekkel. Példának okáért a 16. ábra jobboldali részén egyértelműen látszik, hogy a nagy zajnak köszönhetően a maximum keresés egyáltalán nem egyértelmű feladat.

Viszont ha tévesen állapítjuk meg a forgatás szögét, akkor onnantól kezdve már csak hibás eredményt kaphatunk, hiszen így már az x és y koordinátákra alkalmazott keresztkorreláció nem fog releváns eredményt adni.

Az igazság tehát az, hogy abban a formában, ahogyan a kutatók leírták ezt az algoritmust, nem életképes.

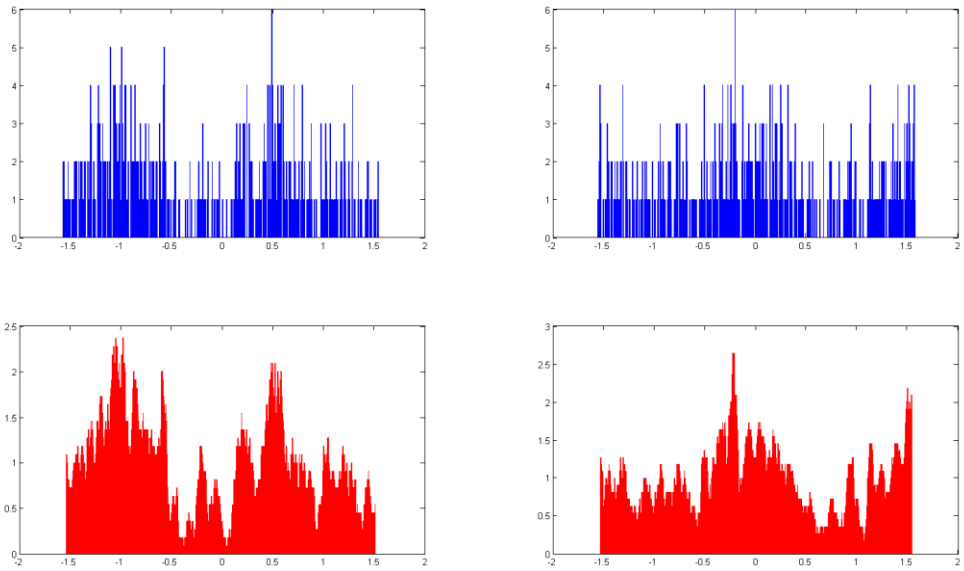
4.6 Javítási javaslatok

Nem adtam fel azonban, hogy az algoritmust valamilyen módon működésre bírjam. A következőkben néhány javítási javaslatomat fejtem ki.

4.6.1 A zaj szűrése

A legelső gondolatom természetesen az volt, hogy ha zajos a szöghisztogram, akkor alkalmazzunk rá valamilyen szűrést.

Ha zajszűrésről van szó, akkor az első próbálkozások egyike mindig az átlagolás szokott lenni, én is így tettem.

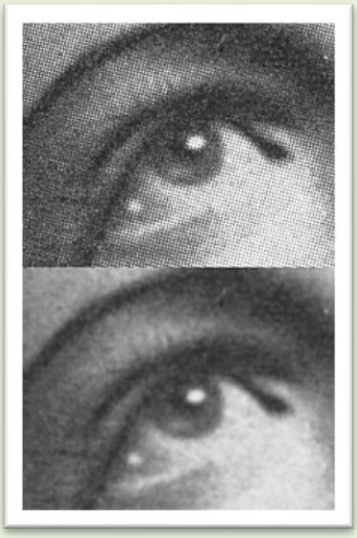


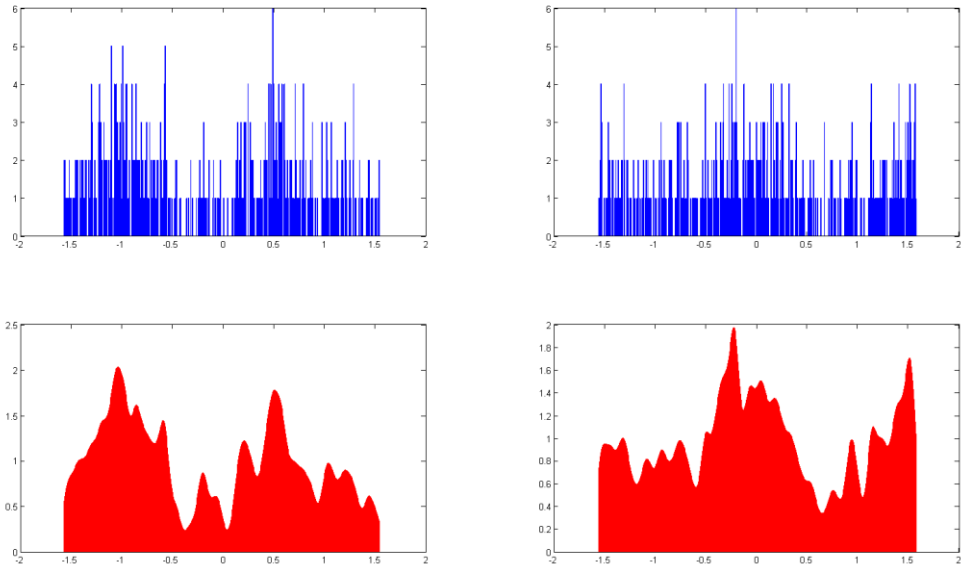
17. ábra - Mozgóablakos átlagolás eredménye

A 17. ábrán az látható, hogy egy mozgóablakos átlagolást alkalmaztam a szöghisztogramon. Az ablak 10 hosszúságú volt. Egyértelműen látszik, hogy a zaj csökkent, ellenben sajnos azt a problémát nem sikerült orvosolni, hogy nem egyértelmű a maximum hely. Azt tapasztaltam, hogy ezáltal megnőtt a szögelfordulás számításának pontossága, azonban a pozíció számítása továbbra is hibás eredményeket szült.

A következő kísérletben Gauss szűrőt alkalmaztam. Az ötlet abból adódott, hogy ránézésre a szöghisztogram közelíthető két egymás után lévő haranggörbével.

Érdekesség: A Gauss szűrőt széles körben használják képfeldolgozás során, mivel hatékonyan szűri a fehér zajt. Az alábbi ábra jól szemlélteti, hogy a felső zajos képet hogyan szűri meg a Gauss szűrő. (http://en.wikipedia.org/wiki/Gaussian_blur)





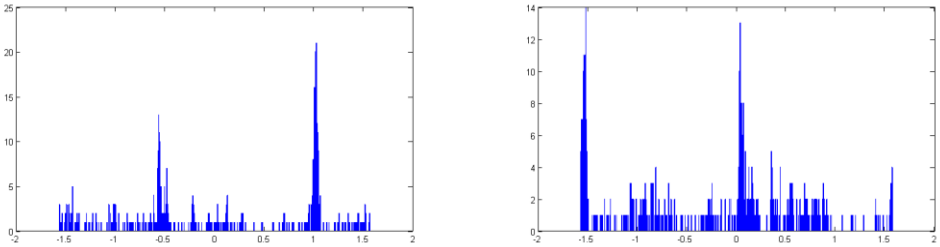
18. ábra - Gauss szűrő alkalmazásának eredménye

A 18. ábra kiválóan szemlélteti a Gauss szűrő hatását. Azt sikerült elérni, hogy a maximumhely már egyértelmű, viszont cserébe jóval kevésbé pontos az ábránk, a szűrő hatására eltolást is szenvedhettek a pontok.

A szöghisztogram szűrése tehát önmagában nem jelent megoldást a problémánkra, ráadásul ha veszteséges szűrészt alkalmazunk, akkor az könnyen többet árthat, mint használ.

4.6.2 A szöghisztogram számítási módszer módosítása

Miután a szöghisztogramban keletkező zaj szűrésével nem jártam sikerrel, már irányba próbálkoztam. Igen jó eredményeket lehet elérni az által, ha a szöghisztogram számítása során nem a szomszédos, hanem távolabbi pontokra számítjuk ki a szögeket. Ekkor ugyanis a távolságmérés hibája arányaiban kisebb lesz, mint két pont távolsága.



19. ábra - Szöghisztogram távolabbi szögekre

Ha a 19. ábrát összevetjük a 18. ábrával, szembetűnő, hogy mennyivel kisebb a zaj. Ebben az esetben az egymástól 20 távolságra lévő pontokkal számítottam a szöghisztogramot. Az eredmény impozáns, a maximumhely így már egyértelmű, ezzel a módszerrel sokat sikerült javítani az algoritmus pontosságán.

4.7 A végítélet

Az algoritmust nagyon szellemesnek találtam, a keresztkorrelációnak valóban nagyon jó a zajtűrő képessége, és nagyon ötletes, hogy a kép forgatásával érjük el, hogy alkalmazható legyen. A megoldás Achilles-sarka egyértelműen a kép elforgatás szöge, ha nem elég pontosan forgatjuk be a két képet, akkor egészen téves eredményre jutunk.

A másik hátránya az algoritmusnak a lassúsága, sajnos a keresztkorreláció számításigényes feladat.

5 A terepszakasz párosító algoritmusok

Az előzőekben vázolt algoritmus alapvetően a falak megtalálására és egymáshoz mérésére apellál. A második vizsgálandó algoritmusnak szándékosan ennek a szöges ellentétét választottam, ebben kifejezetten a kiugró értékek keresése a cél. Az ilyen típusú algoritmusok gyűjtőneve jellegzetes tereppont-alapú lokalizációs algoritmusok.

> **Megjegyzés:** Az angol szakirodalomban erre a feature-based localization kifejezést használják. A feature szó lefordítása nem egyszerű, hiszen rengeteg jelentése van, pl. tulajdonságot, jellemzőt, vonást, jellegzetességet, stb. jelenthet. Ezek viszont önmagukban nem fejezik ki igazán jól a jelentéstartalmat, emiatt választottam a jellegzetes tereppont-alapú lokalizáció kifejezést, ami bár hosszabb, cserébe kifejezőbb.

Számos dolgot tekinthetünk jellegzetes tereppontnak, például a széklábakat, a szoba sarkait, kiszögeléseket, valamilyen lépcsőszerű alakzatot. Ha kültéri alkalmazásban gondolkozunk, akkor ezek a tereppontok lehetnek például fák törzsei, melyeket elég jól közelíthetünk hengerfelülettel.

Az alap gondolat az, hogy az aktuális (*S*) és a korábbi mérés (*R – reference scan*) hasonló jellegzetes pontjait megtalálva kiszámolható a robot elmozdulása és elfordulása. A feladat három nagy problémakörre bontható le:

1. Definiálni kell, mit tekintünk jellegzetes pontnak, majd meg kell találni őket mind az R mérésben (*M – model set*), mind az S mérésben (*D – data set*).
2. Meg kell találni az összetartozó jellegzetes pontokat, azaz olyan párokat $(\breve{M}, \breve{D})$ kell alkotni, melyek első tagja az M halmazból, második tagja a D halmazból való.
3. Meg kell találni azt az optimális transzformációt, amely a $\breve{D}$ pontokat az $\breve{M}$ pontokra képezi, ebből származtatható a pozíció megváltozása, $\Delta p = (\Delta x, \Delta y, \Delta \varphi)^T$

Mind a három feladat már önmagában is kihívást jelent, számos cikk született, amelyik kifejezetten csak az egyikre fókuszál, valamilyen új, esetleg jobb megoldást tálalva.

5.1 A kiválasztott algoritmus

A fent leírt nagy családból én vizsgálódásom tárgyául Kai Lingemann, Andreas Nüchter, Joachim Hertzberg, Hartmut Surmann kutatók által leírt HAYAI (4) (high-speed and yet accurate indoor/outdoor tracking) algoritmust választottam. A miatt esett erre a választásom, mivel az előzőekben vizsgált algoritmusnak az volt az egyik legnagyobb hátránya, hogy igen lassú, a HAYAI viszont kifejezetten sebességre van optimalizálva, és már a nevében is igen gyors futás időt ígér.

5.2 Adatok szűrése

A leírás rögtön azzal kezdődik, hogy milyen szűrért alkalmaztak az adatpontokon. Ez szimpatikus, ezt nagyon hiányoltam az előző algoritmusból. Tapasztalataik alapján Gauss zaj és úgynevezett só és bors zaj terheli a jelet. Az utóbbi sarkoknál fordul elő, ahol a lézernyaláb több felületről visszaverődve jut a szenzorba, hibás adatot eredményezve. Az én méréseimben nem tapasztaltam hasonlót, mivel a mi lézerszkennerünk, a ROD4plus tartalmaz szűrést a többszörösen visszaverődő sugarak ellen. Ami még szintén probléma lehet, ha olyan felületeket ér a lézersugár, amely eltorzítja, pl. egy üvegfelület.

A Gauss zaj elnyomására a következő szűrőt javasolják. Annak következtében, hogy a lézerszkenner köralakban bocsájtja ki a nyalábokat, a forráshoz közelebb lévő pontok sűrűbben helyezkednek el. Azokat az adatpontok, melyek közel esnek egymáshoz egy pontban egyesítjük. A mediánszűrő eltávolítja kieső pontokat azáltal, hogy kicseréli az értékét az őt körülvevő n pont mediánjával. Azonban mielőtt ez a csere megtörténne, kiszámításra kerül az eredeti és az új érték euklideszi távolsága. Annak érdekében, hogy megtartsuk azokat a kiugrásokat, amelyek nem zaj, hanem valós terepviszonyok következtében keletkeztek, csak akkor cseréljük ki a pontot, ha a számított távolság egy adott határérték alatti. (A cikkben 200 cm-t írnak, de én ezt elírásnak tartom, valószínűleg 200 mm-t szerettek volna). Én ezt a határértéket 30 mm-re választottam, mivel a lézerszkenner leírása szerint ± 15 mm-es tartományban lesz a mérési eredmény.

5.3 Jellegzetes tereppontok meghatározása

Mindenekelőtt definiálni kell, mit tekintünk jellegzetes tereppontnak. Ez attól függően változhat, hogy milyen környezet szeretnénk alkalmazni a robotot. Ahhoz azonban, hogy alkalmazhatóak legyenek a lokalizációra, minimálisan a következő követelménynek kell megfelelniük:

- Invariánsnak kell lenniük a forgatásra és az eltolásra.
- Hatékonyan számíthatónak kell lenniük, hogy megfeleljenek a valós idejű elvárásoknak.

A kutatók egy igen gyors és egyszerű megoldást javasolnak a jellegzetes pontok megtalálására, lineáris szűrőket alkalmazva. Céljuk, hogy megtalálják a szélsőértékeket az adatpontok polárkoordinátás reprezentációjában. Ezek a szélsőértékek a Descartes térben sarkaknak és kiugró széleknek fognak megfelelni. A polárkoordinátás alak használata azt is eredményezi, hogy a szűrőket csak egy dimenzióban kell alkalmazni, hiszen fix szögenként mintavételeztük a távolságokat. Ennek következtében fele annyi számításra lesz szükség, mintha külön x és y koordinátákra kéne alkalmazni a szűrőket.

A lineáris szűrőt az alábbiaknak megfelelően definiálják:

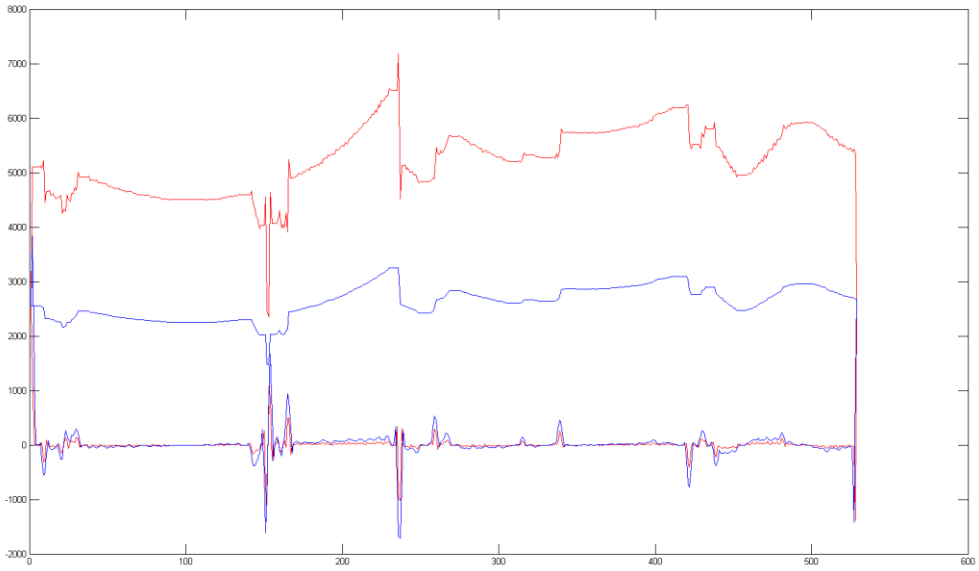
- a mért adatok sora, amelyre alkalmazni kell a szűrőt: $(r_i)_{i\in N}$, amely az aktuális szken $S = ((\varphi_i, r_i))_{i=1\dots N}$ távolság koordinátáiból áll
- A szűrő paraméterei: $\Psi = [\Psi_{-1}\ \ \Psi_0\ \ \Psi_1]$
- A szűrt értékek a következő számítás eredményeként állnak elő:

$$r_i^{\Psi} = \sum_{k=-1}^{1} \Psi_k r_{i+k}$$

A jellegzetes tereppontok megtalálásához három szűrőn keresztül kell átszűrni az adatokat:

1. Először egy élesítő szűrőt alkalmazunk, amely kiemeli a számunkra lényeges részeket, azaz a szélsőértékeket: $\Psi_1 = [-1\ 4\ -1]$
2. Kiszámítjuk a jel deriváltját: $\Psi_2 = \left[-\frac{1}{2}\ 0\ \frac{1}{2}\right]$
3. Elsimítjuk a deriváltat a nullátmenetek könnyebb detektálásához: $\Psi_3 = [1\ 1\ 1]$

Bár a szerzők nem térnek ki rá, én érdemesnek tartom megjegyezni, hogy a második lépésben nem a hagyományos differenciálást használjuk, amely a szomszédos pontok különbségét számolja, hanem a kettővel mellette lévővel számol. Ennek egyrészt az az előnye, hogy kevésbé érzékeny a zajra, másrészt a szimmetrikus Ψ-nek köszönhetően valóban az aktuális pontban kapjuk meg a derivált értékét.



20. ábra - A HAYAI szűreinek alkalmazása

A fenti ábrán látható az érmérési és számítási eredményeim ábrázolása. A felső kék a medián szűrőn átesett mérési adatsor. A felette lévő piros az élesített adatsor, ezen rögtön látszik ennek hátránya is: jól láthatóan a zajt is erősíti a szélsőértékek mellett. Az alsó kék adatsor a deriváltat ábrázolja, a piros pedig annak simított változatát.

Az elgondolás szemmel láthatóan jón működik, a végeredményül kapott adatsorban valóban csak ott van szignifikáns eltérés a nullától, ahol jellegzetes tereppontok vannak. A nullátmenetek megkeresésével megkapjuk a jellegzetes tereppontok halmazát.

A nullátmenetek keresésére a szerzők nem közölnek optimális megoldást, az én javaslatom a következő:

5-1. kódrészlet - Nullátmenetek keresés

```
s = sign(rs_d);
t = filter([1 1],1,s);
zc_s = find(t==0);
```

A megoldás egyszerű és elegáns: signum függvényt generálok, majd alkalmazok rá egy olyan szűrőt, amely összead minden szomszédos pontot. Az így kapott függvény pont ott lesz nulla, ahol az eredeti adatsorban nullátmenet volt, hiszen ekkor lesz egymás után a szignum függvényben -1 és +1.

A jellegzetes pontok generálását elvégezzük mind az M, mind a D halmazon.

5.4 Jellegzetes pontok párosítása

Az összetartozó tereppontok párosítása egyáltalán nem könnyű feladat, klasszikus megoldása az úgynevezett RANSAC (5) algoritmus. Igen széles irodalommal rendelkező, jól kidolgozott algoritmus, amely a leírások alapján igen jó eredményt ad. Hátránya, hogy egyáltalán nem nevezhető gyorsnak, így a HAYAI megalkotói valami egészen mást javasolnak helyette.

Az ötlet, hogy építsünk egy mátrixot a lehetséges párokból, a mátrix elemei pedig a következő képlet állítsa elő (az m_i és d_i értékek az M illetve a D halmazok elemeinek koordinátái polár-, ill. Descartes-koordinátarendszerben):

$$\mathrm{dist}(\boldsymbol{m}_i, \boldsymbol{d}_j) = \sqrt{\omega_1(m_i^{\phi} - d_j^{\phi})^2 + \omega_2(m_i^{r} - d_j^{r})^2} + \sqrt{\omega_3(m_i^{x} - d_j^{x})^2 + \omega_4(m_i^{y} - d_j^{y})^2} + \Theta(\boldsymbol{m}_i, \boldsymbol{d}_j)$$

Jól látszik, hogy alapvetően heurisztikus megközelítésről van szó, hiszen olyan értékeket adunk össze, amelyek különböző koordinátarendszerekből valók. Viszont a gondolatmenet tisztán látszik, valamilyen távolsággal kapcsolatos fogalmat vezet be a szerző. Érdemes azonban észben tartani, hogy ennek az értéknek egyébként nincs semmilyen fizikai tartalma.

A képlet végén található tag arra szolgál, hogy csak ugyan olyan típusú pontokat hasonlítsunk össze. Ugyanis az önmagában, hogy a derivált nulla, még három dolgot is jelenthet: maximum értéket, minimum értéket, vagy szingularitási pontot. Fontos, hogy ezek közül csak ugyanolyanokat próbáljunk párosítani, az az ne párosítsunk maximális értéket egy minimális értékkel. Ennek megfelelően a képlet végi teta értéke csak akkor nem végtelen, ha m_i és d_i ugyan olyan típusú.

Eddig a pontig a HAYAI leírása nagyon egyértelmű és jól követhető, semmilyen problémám nem adódott az implementálás során. Ellenben itt van egy nagyon nagy hiányosság a cikkben, nevezetesen nem igazán írja le, hogy az így kapott mátrixból hogyan kapható meg a párosítás. Abból, hogy ezt egyfajta hibafüggvényként definiálja, az ember azt sejti, hogy valamilyen minimalizációra lesz szükség.

Hosszabb kutatómunka után találtam egy leírást (6), amely a következő megoldást nyújtja erre a megoldásra:

„After dissimilarity matrix is constructed, at each step the minimum value element of this matrix is chosen, its corresponding row and column are omitted and correlated features are matched, until all rows and columns of dissimilarity matrix are omitted or the minimum value of elements exceeds a maximum matching distance."

Tehát azt javasolja, hogy egyszerűen mindig a legkisebb elemet keressük a mátrixban, így előállnak a párok.

5.5 Pozíció számítása

Az összetartozó párok megtalálása után az új pozíció számítása következik. A szerzők itt egy nagyon szép levezetést és logikusan felépített levezetést írnak le. Én csak a végeredményüket közlöm, mivel dolgozatom szempontjából inkább ez a lényeges, ellenben mindenkinek, aki szereti a szép bizonyításokat, javaslom elolvasásra a levezetést.

$$\Delta\theta = \arctan\left(\frac{\sum_{i=1}^{N}(\hat{m}_i^x \cdot \hat{d}_i^x + \hat{m}_i^y \cdot \hat{d}_i^y)}{\sum_{i=1}^{N}(\hat{m}_i^y \cdot \hat{d}_i^x - \hat{m}_i^x \cdot \hat{d}_i^y)}\right)$$

$$\underbrace{\begin{pmatrix}\Delta x\\ \Delta y\end{pmatrix}}_{=\Delta \boldsymbol{t}} = \begin{pmatrix}\check{c}_m^x\\ \check{c}_m^y\end{pmatrix} - \underbrace{\begin{pmatrix}\cos\Delta\theta & \sin\Delta\theta\\ -\sin\Delta\theta & \cos\Delta\theta\end{pmatrix}}_{=\boldsymbol{R}_{\Delta\theta}} \cdot \begin{pmatrix}\check{c}_d^x\\ \check{c}_d^y\end{pmatrix}$$

Jól látszik, hogy miután megtaláltuk az összetartozó párokat, az elmozdulás és az elfordulás számítása könnyen és gyorsan adódik.

5.6 Mérési eredmények

Ahogyan az már a 20. ábra eredményeiből is látszik, a valós nullátmenetek megtalálását nehezíti a jelet terhelő zaj, nem könnyű megkülönböztetni azt, hogy csak egy hiba miatt volt nullátmenet, vagy valóban szélsőértéke van ott a függvénynek. Tehát a jellemző tereppontok megtalálása sikerült, viszont több hamis eredmény is került a halmazba.

Ezekkel az eredményekkel a fent leírt módon sajnos nem sikerült a párosítás megvalósítása, és mivel teljesen heurisztikus megoldásról van szó, így nehéz lett volna tudományos magyarázatot adni arra, hogy miért nem működik. Ez a terület még további kutatást igényel.

A (6)-os cikkben erre azt írják, hogy ez az algoritmus inkább nagy távolságú, kültéri mérésekben ad jó eredményt, könnyen lehet, hogy az én beltéri méréseimre emiatt nem működött igazán jól.

Fontos azonban kiemelni, hogy a futási sebessége nagyságrendekkel jobb, mint az elsőként bemutatott algoritmus, szóval ígéretes lehet valamilyen módosított formában kísérletezni a megvalósítással.

6 Összefoglalás

Dolgozatomban első felében bemutattam azt a keretrendszert, amelyben a lézeres távolságszkenner-alapú lokalizációs algoritmusokat vizsgáltam, majd részletesen bemutattam magát a konkrét szenzort, legfontosabb paramétereit értékeltem. A dolgozatom második felében pedig bemutattam a lokalizációs algoritmusokat, azok tulajdonságait, implementációs nehézségeit, valamint javaslatokat adtam a megvalósításhoz.

Azt gondolom, hogy azok a kiegészítések, javaslatok, bemutatott problémák, és a rájuk adott megoldási kísérletek jelentik a dolgozatom legnagyobb értékeit, emiatt bátran ajánlom olyanoknak, akik most szeretnénk elkezdeni foglalkozni ezzel a területtel.

7Irodalomjegyzék

1. **Wikipedia.** LIDAR. [Online] 2012. [Hivatkozva: 2012. 10 24.] http://en.wikipedia.org/wiki/LIDAR.

2. **electronic, Leuze.** *Technical Description.* [Dokumentum] 2007.

3. **Gerhard Weiß, Christopher Wetzler, Ewald von Puttkamer.** *Keeping Track of Position and Orientation of Moving Indoor Systems by Correlation of Range-Finder Scans.* Kaiserslautern : University of Kaiserslautern, 1994.

4. **Kai Lingemann, Andreas Nüchter, Joachim Hertzberg, Hartmut Surmann.** *High-speed laser localization for mobile robots.* Osnabrück, Germany : University of Osnabrück, 2005.

5. **Marco Zuliani.** *RANSAC for Dummies.* Santa Barbara : Marco Zuliani, 2012.

6. **A. A. Aghamohammadi, H. D. Taghirad, A. H. Tamjidi, and E. Mihankhah.** *Feature-Based Laser Scan Matching For Accurate and High Speed Mobile Robot Localization.* hely nélk. : K. N. Toosi University of Technology, 2010.

7. **Ya-Fu Peng, Chih-Hui Chiu, Wen-Ru Tsai, and Ming-Hung Chou.** *Design of an Omni-directional Spherical Robot: Using Fuzzy Control.* Hong Kong : Proceedings of the International MultiConference of Engineers and Computer Scientists, 2009.

8. **Martin Hall, Mark Towfiq.** Windows Sockets. [Online] 1992. [Hivatkozva: 2012. 05 18.] http://www.sockets.com/winsock.htm#WhatIs.

9. **Jones, M. Tim.** Five pitfalls of Linux sockets programming. [Online] 2005. [Hivatkozva: 2012. 05 18.] http://www.ibm.com/developerworks/linux/library/l-sockpit/.

10. **Jefri Efendi Mohd Salih, Profesor Dr Sazali Yaacob,Prof. Madya Dr Mohd Rizon Mohd Juhari.** *Omni-Directional Mobile Robot with Mecanum Wheel.* Malaysia : Kolej Universiti Kejuruteraan Utara Malaysia, 2005.

11. **J. Borenstein, H. R. Everett, L. Feng.** *Where am I?* Michigan : University of Michigan, 1996.

12. **Iagnemma, Martin Udengaard and Karl.** *Design of an Omnidirectional Mobile Robot for Rough Terrain.* Cambridge : Massachusetts Institute of Technology, 2007.

13. **C. Douglass Locke, Ph.D.** *POSIX and Linux Application Compatibility Design Rules.* 1.0. 2006.