

Óbudai Egyetem

Kandó Kálmán Villamosmérnöki Kar

Műszertechnikai és Automatizálási Intézet



TUDOMÁNYOS DIÁKKÖRI DOLGOZAT

AUTONÓM MŰKÖDÉSŰ MOBIL ROBOT

Szerző(k): Pintér Zoltán

villamosmérnök BSc. szak, III. évf.

Róth Kálmán

villamosmérnök BSc. szak, III. évf.

Konzulens: Dr. Kucsera Péter

egyetemi docens

Budapest, 2013

Bevezető

A dolgozat témája:

A dolgozat célja ismertetni egy autonóm, önműködő, mozgásra képes robot felprogramozása, mely magától képes kikerülni az előtte felmerülő akadályokat, mozgását koordinálni. A feladat célja továbbá a megvalósításhoz szükséges eszközök, rendszerek megismerése.

Kiindulási alap:

A projekt alapja a jól felépített hardverstruktúra, mely jó minőségű ipari eszközöket tartalmaz. A projekt igazi kihívása, hogy egy „fekete” dobozt kaptunk, melynek hardverfelépítését meg kellett ismernünk, továbbá, hogy megismerkedjünk a Phoenix Contact PLC programozás technikájával.

A TDK dolgozat az iskolai mobil robot projektre épül, annak funkcióinak bővítése és magasabb szintű megismerése céljából.

Munkásságunk:

A mobil robot projekt a tantervi iskolai projekthez kapcsolódik, és ezen tantárgy keretében ismerkedtünk meg a robottal. A mobil robot minden évben más-más hallgatócsoport feladata volt, így minden évben történt rajta fejlesztés, finomítás. Amikor a robotot kézhez kaptuk, akkor szoftveresen semennyi intelligenciával sem rendelkezett. Az előző évfolyamok szoftveres fejlesztései nem álltak rendelkezésünkre, így egy nagyon régi vezérlőprogram tanulmányozásán keresztül jutottunk el a robot felépítésének a megismeréséhez. Ebben az egyetlen működőképes program a DC szervó hajtás vezérlő programrész volt. Ennek a programnak a részletes megismerése, értelmezése is a feladat része volt, hiszen enélkül nem lehetett volna továbblépni. Mivel a robot működésképtelen volt, így feladatunk volt az akadályfelismerő algoritmus megírása. Ez a feladat önmagában nem jelentette volna a robot fejlődését, így az újdonság a 3D-s kamera felinstallálása, a szoftver megírása volt. A gyakorlati tesztelés és fejlesztés során kiderült, hogy a robot teljesen védtelen a mélyedésekkel, lefelé irányuló lépcsőkkel szemben, így kifejlesztésre került egy egyszerű lépcsőérzékelő, ami a PLC I/O portjára egy relén keresztül jelzi vissza az akadály jelenlétét.

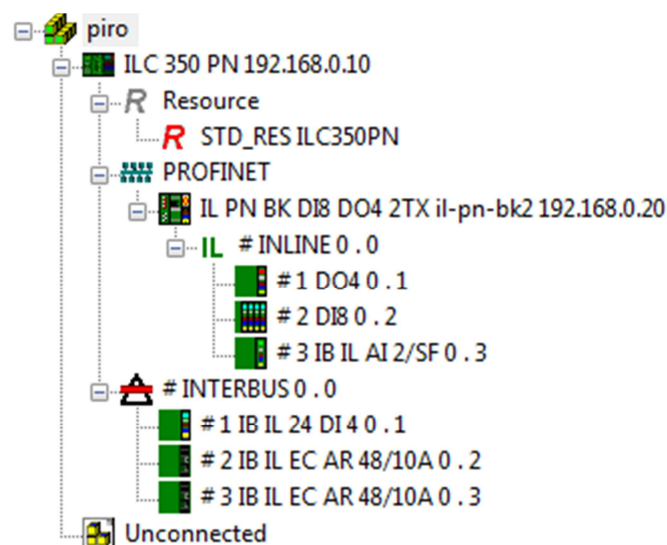
Hardver struktúra:

A mobil robot projekt egy kiépített hardver struktúrára épül, mely tartalmaz minden olyan egységet, ami a robot mozgásához, meghajtásához szükséges. A robot logikai vezérlését egy Phoenix Contact ILC 350 PN típusú PLC végzi. A PLC-hez Profineten keresztül kapcsolódnak a diszkrét és folytonos I/O kártyák. A robot mozgásáért 2 db szervomotor felelős, melyek a PLC által vezérelt Interbus-on keresztül az erre a célra beszerelt szervó meghajtó modul által vannak vezérelve.

A rendszer tápellátásáért egy 24 V-os kapcsolóüzemű tápegység felel, mellyel párhuzamosan a rendszer egy 12 Ah kapacitású akkumulátoron keresztül is táplálható, valamint az akkumulátor töltése is megoldható, az akkumulátor saját töltőelektronikájával.

A PLC programozásához egy robotra telepített Phoenix Contact ipari PC-t használtam, melynek további funkciója a kezelőfelület, illetve a PLC közötti kapcsolat megvalósítása.

A PLC és PC közötti kapcsolatot egy ipari Phoenix Contact switch valósítja meg, mely továbbá lehetőséget ad a PLC másik számítógépen keresztüli programozására is.



1. 1. ábra

Buszstruktúra felépítése

A mobil robot hardverstruktúrájában szereplő perifériák (I/O blokkok és a motorvezérlők) ipari terepi buszrendszeren kapcsolódnak a Phoenix Contact PLC-hez, így fontosnak találom a továbbiakban ezen buszrendszereket ismertetni.

A folyamatirányításban a technológia fejlődésével, összetettebbé válásával együtt jár a perifériák számának növekedése, minél bonyolultabb egy irányítandó technológia, annál több digitális és analóg jelekkel megfeleltethető folyamatjellemző lekezelése szükséges. Az iparban ezek a folyamatjellemzők legtöbbször fizikailag távol vannak a vezérlő egységtől (pl. PLC), mégis elvárás, hogy a jellemzők lekezelése konzisztens módon, megfelelő időkritériumok mellett történjen meg. Gazdaságossági szempontból is fontos, hogy hogyan lehet a termelést gazdaságossá tenni, hogyha az egyre bonyolultabb technológia egyre több jel lekezelését teszi szükségessé, ezzel párhuzamosan a vezetékezési költség növekedését is előíranyozza. Analóg jelek esetében speciális kábelek alkalmazása lehet szükséges, továbbá a szerelési költségek, illetve a nagyobb meghibásodási valószínűségből adódó többletköltségek előíranyozzák más megoldások keresését.

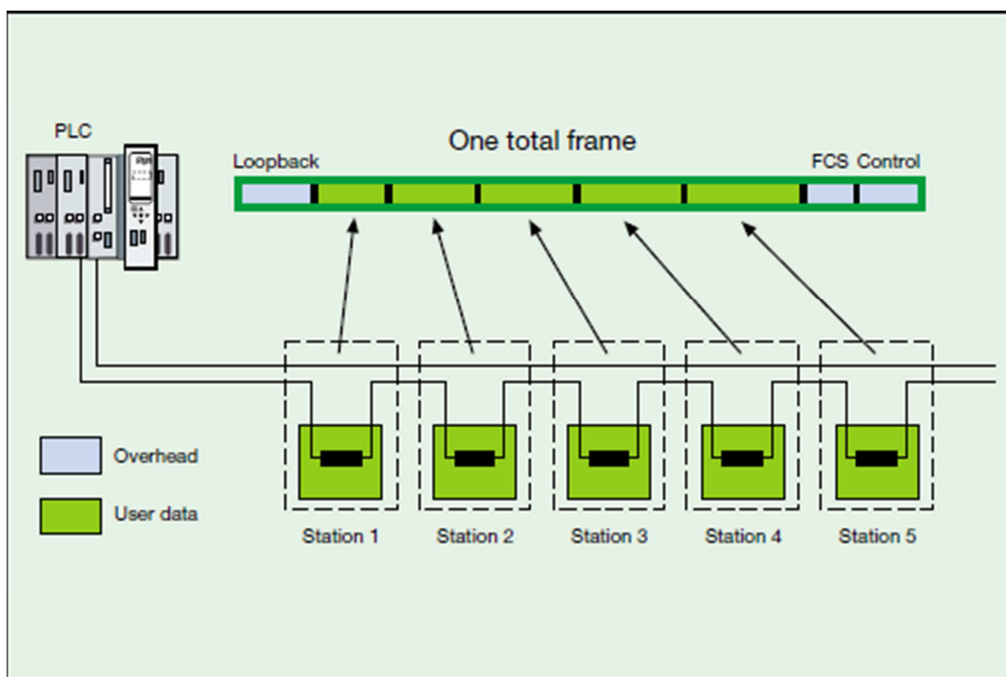
Az egyik ilyen megoldás a terepi busz rendszer kialakítása. Ennek lényege, hogy a párhuzamosított kábelkötegeknek a hosszát lecsökkentik, és az azokat összefogó decentralis perifériákat egy buszrendszerre fűzik fel. Ennek előnye a kábelköltségek csökkenése, illetve a kisebb meghibásodási valószínűség, továbbá, hogy lecsökken az analóg jelekre ható zavarok támadáspontjainak a száma. Ez utóbbi kritikus fontosságú jelek esetében elengedhetetlen. Az ilyen megoldás lehetővé teszi egy általános buszstruktúra felvázolását, mely a buszprotokoll segítségével könnyen átkonfigurálható, anélkül, hogy helyi szinten át kelljen rendezni a rendszert. Fontos továbbá a rendszer jeleinek a könnyű monitorozása, felügyeleti rendszerekbe való beintegrálhatósága is. Általános elvárás továbbá, hogy ne csak egyféle eszközgyártó termékeit lehessen buszba szervezni, hanem egy gyártó független terepi buszos alkalmazást lehessen kialakítani.

Terepi buszrendszerből többféle létezik, és ezek közül több szempont alapján lehet mérlegelni, hogy melyik kialakítás a megfelelő. Terepi buszos hálózatokban két alapvető adatfolyam létezik, az egyik a folyamatjellemzőhöz kapcsolható (process data) és a paraméterezéshez kapcsolódó adat (parameter data). Előbbi ciklikus kis adatmennyiségű, utóbbi aciklikus (alkalomszerű) nagyobb adatmennyiségű. A mai terepi buszrendszerekkel kapcsolatban elvárás, hogy 5 ms alatt egy teljes buszlekérdezés megtörténjen, így biztosítható a folyamatjellemzők aktualitása, és determinisztikussá válik a busz viselkedése, ami a rendellenességek kiszűrése miatt fontos.

Az előbbiekhöz kapcsolódik a kódhatékonyság fogalma, melyet úgy tudunk könnyen leírni, hogy ha egy teljes busztranzakcióban az átvitt hasznos adatot viszonyítjuk az összes átvitt adatmennyiséghez. A kódhatékonyságot leginkább a protokoll kommunikációs módszere befolyásolja. Az üzenetorientált kommunikációs protokollok esetében minden egyes adatküldés alkalmával az adatkapcsolati réteg fejléccel látja el a hasznos adatot, így egy alacsony kódhatékonyságú üzenet jön létre. Ha az üzenetorientált rendszerben ciklikusan történik az adatlekérdezés, akkor a busz kódhatékonysága átlagosan alacsony lesz. A nem ciklikusan küldött paraméteradatok szempontjából, melyek nagyobb adatmennyiségűek a kódhatékonyság javul, hiszen ugyanaz a fejrész nagyobb mennyiségű hasznos adatra vonatkozik. A másik gyakran használt módszer az adatösszegzéses keretképzési módszer. Ebben az esetben az adatkapcsolati réteg az egyes eszközök (pl. érzékelő és beavatkozó szervek) által kiküldött keretet rendezi egy üzenetté, így minél több eszköz helyezkedik el a buszon, annál hosszabb lesz ez az üzenet, viszont a hozzá tartozó fejléc hossza változatlan marad, tehát a kódhatékonyság megnő. Előnye ennek a protokollnak, hogy minden eszköz azonos adathosszúságú csomagot tesz az üzenetbe, így az üzenet elküldéséhez szükséges idő a fejrész kivételével arányos az eszközzámmal, így determinisztikus kommunikáció valósul meg, ami az időzítések kalkulációja szempontjából fontos.

Interbus protokoll:

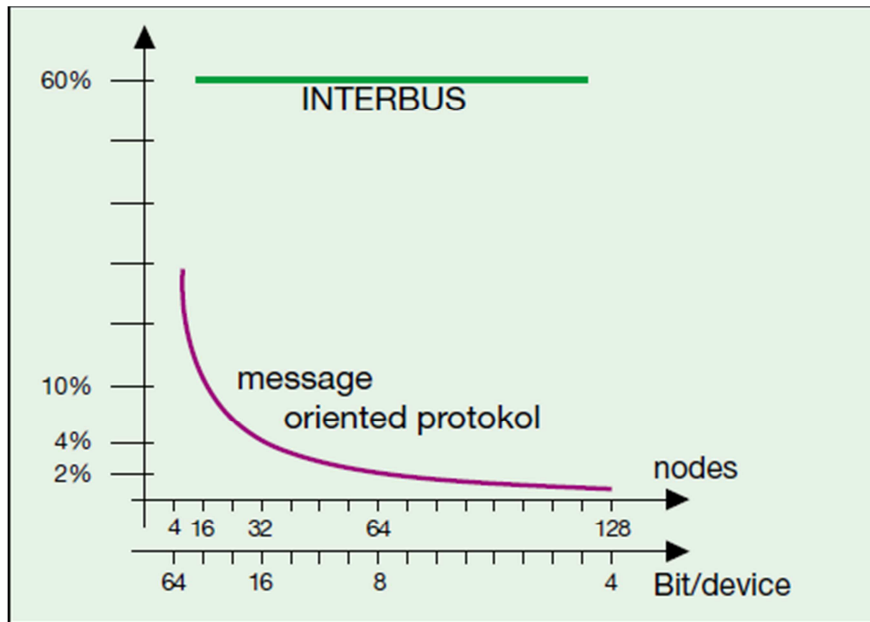
Az előbbi szempontok szerint az egyik legnagyobb kódfatékonyságú terepi busz protokoll az Interbus protokoll. Ez a protokoll soros gyűrű topológiát alkalmaz, és egyedülálló módon valósítja meg a keretösszegzésen alapuló üzenetképzés módszerét.



1.2. ábra

Interbus keretképzés

A fenti ábrán látható a keretösszegzésen alapuló üzenetszervezés modellje. A buszt a Master modul vezérli és a Masterből kimenő keret sorban halad végig az egyes Slave eszközökön, és az eszközök a saját időzésükbe behelyezik az adatukat. Az adatátvitel leellenőrzésére 16 bites CRC kódot használnak.



1.3. ábra

Kódhatékonyság alakulása

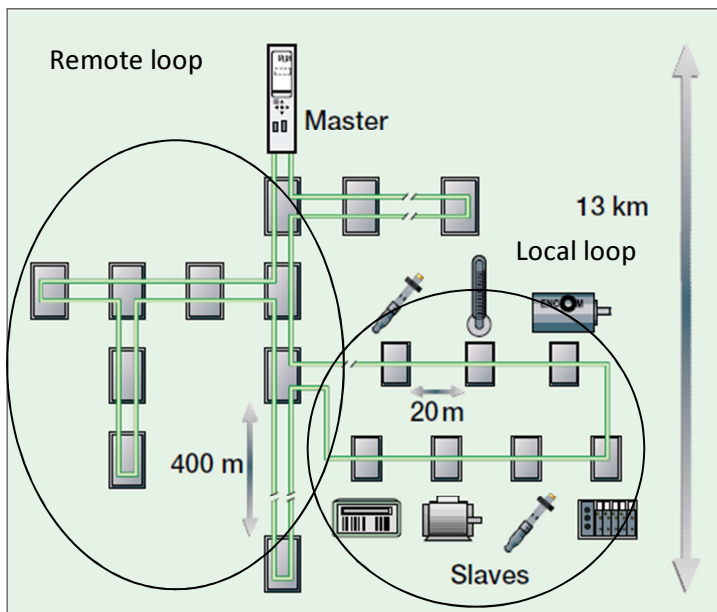
Az 1.3. ábrán látható, hogyan alakul az üzenetorientált és az Interbus protokoll esetében a kódhatékonyság. Nagy kiterjedésű hálózatokban az átvitt bitek 90 %-a is fejrész bit lehet, ami

meglehetősen leterheli

az átviteli közeget, és

nehezíti a busz ciklusidejének a meghatározását. A fenti ábrán látható jó bithatékonyság az Interbus PCP (Peripherals Communication Protocol) közeghozzáférést vezérlő protokolljának köszönhető. Számomra is fontos ennek a protokollnak az ismerete, hiszen, mint a buszstruktúrából látható, a szervó vezérlő modul is Interbus-on keresztül kapcsolódik a PLC-hez, illetve szoftveres szinten is a PCP függvényeken keresztül van vezérelve a szervó vezérlő. A PCP protokollnak az a feladata, hogy a buffer regiszterbe írt adatot sorozatban beleillessze abba az időzésbe, amit a Master a buszon megcímez az eszköz számára. Mivel nem biztos, hogy az elküldendő adat egy időzésbe belefér, így a PCP szétbontja azt bájtokra és FIFO szerűen a következő buszciklus megfelelő időzésébe kiteszi a buszra a fennmaradó részt.

A fizikai kialakítást tekintve a busz két hurokra osztható fel. Egy távoli elérésűre és egy onnan lecsatlakozó helyi hurokra.



1.4. ábra

Tipikus gyűrűs Interbus topológia

Az 1.4-es ábrán látható a két tipikus gyűrűs kialakítás. A távoli elérési hurokra van felfűzve a buszillesztők többsége, melyek terminálként funkcionálnak. Feladatuk a jelkondicionálás, illetve róluk kisebb helyi hurkok ágaztathatóak le. Két terminál között maximum 400 méteres távolság lehet, a teljes hurok hossza pedig 13 km. A kommunikáció sebességét illetően 500 kbps baud rate érhető el, a felfűzhető I/O-k száma pedig 4096-ban maximalizálva van.

Normál üzem közben is előfordulhat, hogy az aktív gyűrű megszakad. Ennek a detektálására szolgál a „Loop-back word”, amit a gyűrűn indított tranzakció során a Master küld ki, és abban az esetben, hogyha a vételi hurkon ezt nem kapja vissza, akkor következtetni lehet a hurok felnyílására. Az Interbus protokoll által garantált adatátviteli idő a térben egymástól nagyobb távolságra elhelyezett eszközök esetében is garantálja a fix és kiszámítható késleltetést, illetve a konzisztens, egy beolvasási ciklushoz tartozó adatok hibamentes átvitelét.

Profinet IO:

A Profinet IO protokoll a hagyományos Ethernet fizikai rétegét használja (villamos paraméterek és fizikai interfészek), ám kommunikációs csatornáját tekintve úgy lett megtervezve, hogy tudjon kommunikálni PLC-kel és egyéb ipari logikai eszközökkel. A protokoll kidolgozására elsődlegesen gazdasági megfontolásokból került sor, hiszen az alap időfüggő TCP/IP folyamatvezérlő protokoll időkritériuma a CSMA/CD ütközésfigyelő protokollal együtt nem alkalmas időkritikus rendszerek folyamatjellemzőinek megfelelő továbbítására. Az Ethernet technológia szélesebb körű elterjedésével és az ahhoz kapcsolódó hálózati eszközök árának csökkenésével megnyílt a lehetőség egy szabványos ipari Ethernet

hálózati szabvány létrehozására. Mint említettem, az alap Ethernet az ütközésérzékelős közeg hozzáférési protokoll miatt alkalmatlan volt ipari felhasználásra, a véletlenszerűségéből adódóan prioritás szintű irányítás nem volt megvalósítható. Gyakorlati példa lehet egy vészleállító nyomógomb és egy E generációs hőmérséklet távadó esete, amikor a nyomógombhoz csatolt buszillesztő I/O modul üzenetét felülírja egy nem kritikus alrendszerbe tartozó hőmérséklet távadó üzenete. Ütközés esetén nem előre kiszámítható, hogy mikor fog egyik-másik üzenet érvényre jutni a buszon, így a vészleállításhoz szükséges üzenet akár másodperceket késhet, így emberi életet is veszélyeztethet. Ebből adódóan a hagyományos TCP/IP kommunikációs csatorna mellé létre kellett hozni valós idejű elérést biztosító csatornákat is. A valós idejű elérésért felelős egyik csatorna a „Real Time”, melyen 10 ms-os késleltetéssel az üzeneteknek érvényre kell jutniuk a vevő oldalon. Az előző példára építve a nyomógomb megnyomását követő 10 ms-on belül a központi vezérlőnek tudomást kell szereznie erről az eseményről. Gyakorlatban ezt a csatornát ciklikus adatcsere céljából használják, ahol a folyamatjellemzők konzisztenciája és érvényessége kritikus fontosságú. Ez a garantált 10 ms-os időkorlát nem gyorsan változó diszkrét ki- és bemenetek, illetve analóg jellemzők esetében megfelelő, viszont hajtásvezérléshez illetve olyan alkalmazásokhoz, ahol nagy frekvenciás jelekkel dolgoznak ez az időkritérium nem megfelelő. Ez utóbbi jelek is a ciklikusan küldött folyamatjellemzők kategóriájába tartoznak, az ezekhez rendelt csatorna az „Isochronous Real Time”, mely nagyon gyors elérést tesz lehetővé, így kevesebb, mint 1 ms elteltével érvényre kerülnek a küldött adatok. A hagyományos TCP/IP alapú csatornát az aciklikus nagyobb mennyiségű, nem időkritikus paraméter adatok kézbesítésére (írás/olvasás) használják. Ahhoz, hogy nagyobb rálátásunk legyen az egyes csatornákról érkező üzenetek kezeléséről, érdemes megvizsgálni egy Real Time keretet:

Preamble	MAC DA	MAC SA	VLAN	Type	Data	CRC
8 byte	6 byte	6 byte	2 byte	2 byte	n byte	4 byte

A közeghozzáférést tekintve, egy buszciklus felbontható Profinet-es csomagoknak dedikált időszekre és nyílt időszekre (pl. TCP/IP kereteknek). Minden ciklus elteltével a bufferelt

üzeneteket a hálózati irányító eszközök (pl. switchek) megvizsgálták és a „VLAN taget” tartalmazó kereteket előbb továbbítják a hagyományos TCP/IP csomagokkal szemben a megfelelő portra, továbbá a „VLAN tagen” belüli prioritásszint is dominál versengés esetén.

Szervo vezérlés

Interbus-ra van felfűzve a 2 db szervomotor meghajtó egysége.

Ezeknek a típusa : IL EC AR 48/10 A.

Ez a modul képes a motor tengelyének pozíció, nyomaték és sebesség szabályozására. A jelen alkalmazásban direkt módon kell vezérelni a szervo motort vezérlő szavak segítségével. A szervo vezérléséhez az ellenőrzőjeles visszacsatolást a motor tengelyén elhelyezett enkóder adja.

Ahhoz, hogy a szervomotort tudjuk vezérelni először is fel kell konfigurálni az előbb említett motorvezérlőt. A konfigurálást a következő oldalakon található gráf szemlélteti.

A motorvezérlést és a szervo modul inicializálását egy közös blokk hajtja végre, így a kezdeti feltételvizsgálat a meghajtó inicializálását vizsgálja. Ha a modul már inicializálva van, akkor a vezérlő blokk a motor sebességvezérlésre ugrik tovább, amit a következő oldalakon fogunk ismertetni. Az első beállítandó paraméter a motor típusának megadása. A 2-es konfigurációt kellett kiválasztani, ezzel megadjuk, hogy DC motort szeretnénk vezérelni, illetve hogy az INC1-es bemenetre (X6) van bekötve az inkrementális enkóder. A modul az inkrementális enkóder impulzusaiból belsőleg számítja ki a forgási sebességet, és ezt továbbítja a PLC felé Interbus-on keresztül. A motor nyomatékát és áramfelvételét a modul által mért áramfelvételtől számítja/nyeri ki a motor vezérlő.

Az enkóder felbontás beállításával az enkóder típusának függvényében tudjuk beállítani az egy teljes elforduláshoz tartozó impulzusok számát, ez a mi esetünkben 65536. Az enkóder típusának megadása alatt a kimeneti jel villamos szimmetriáját értem, ebben a pontban

megadjuk, hogy aszimmetrikus (földhöz képest értett), vagy szimmetrikus differenciál jeleket küld-e az enkóder tárcsa. Ez a jelenlegi konfigurációban aszimmetrikus.

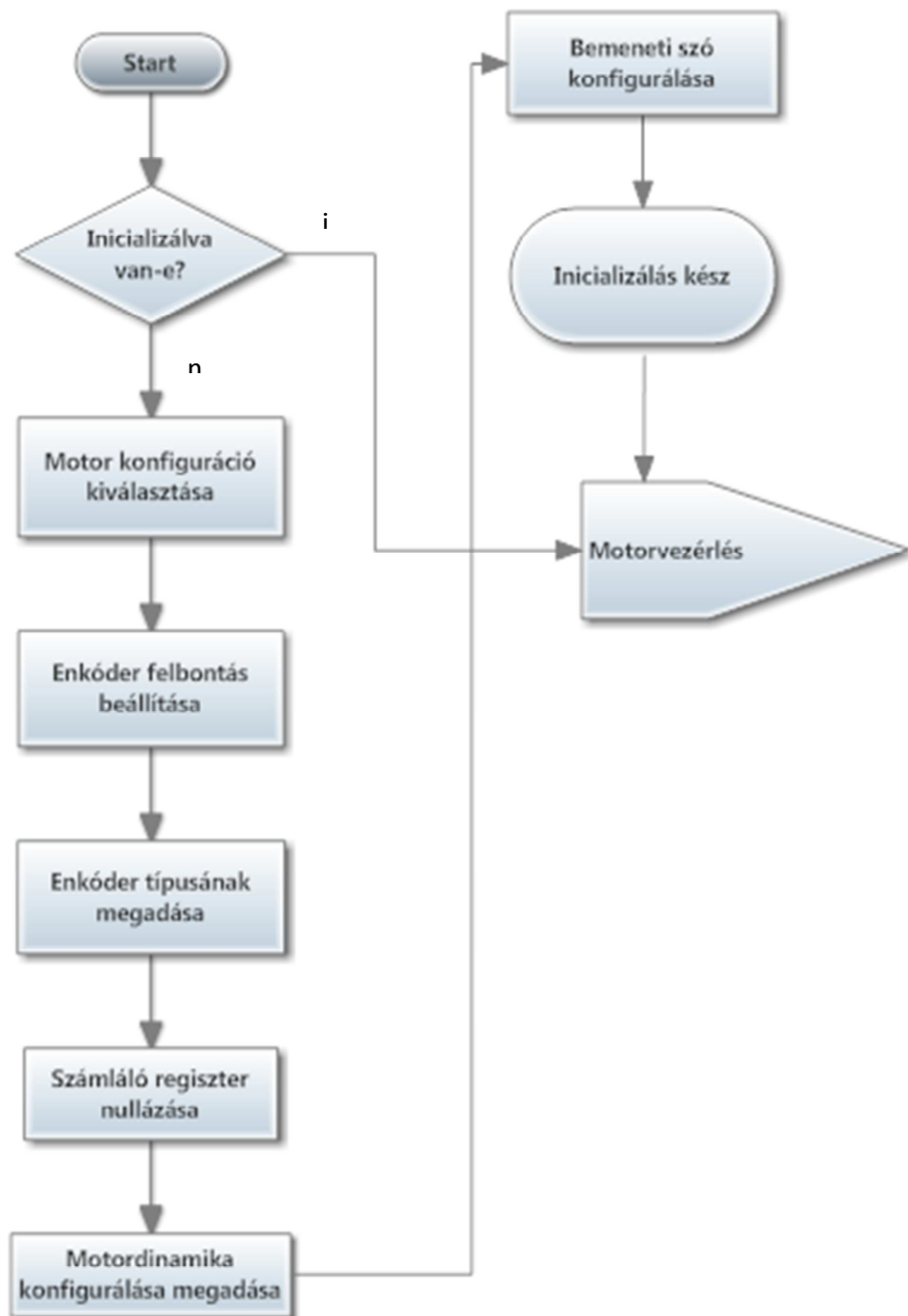
A motorvezérlő engedélyezése előtti fontos lépés az enkóder impulzusokat számláló regiszter törlése. Ezt követően a motor dinamikus jellemzőjének és az Interbus-on a PLC felé visszaküldött adat formátumozása van hátra. A robot megvalósított üzemmódjaiban a motorvezérlő sebességszabályozás üzemmódban működik, és ehhez az üzemmódhoz kapcsolódóan megadható, hogy a szabályozott jellemző milyen gyorsan változzon. Ez a változás $\Delta w/\Delta t$ formában adható meg, és a mi beállításunkban a Δt 1 s-ra adódott, ami egy nagy integrálási idővel rendelkező PID szabályozást eredményez. Sebesség szabályozott üzemmódban a fékezési meredekség beállításánál megadhatjuk a sebesség gradienst, ami a sebességcsökkenést írja le. Ez jelen esetben a maximumra, 1000000 rpm-re lett beállítva, ami erőteljes lassulását eredményezi a robotnak.

A szervó inicializálás legutolsó pontjaként megadandó, hogy milyen adatformátumban kapjuk vissza az aktuális pozícióját a motor tengelyének. Ez az érték szukcesszív mivel inkrementális az enkóderünk, tehát abszolút helyzetazonosításra csak úgynevezett homeing után lenne alkalmas, viszont sebességet két kiolvasott értékből már lehet számolni. A visszkapott pozícióérték az IN bemeneti szó 2. bájtjától kezdődően van elhelyezve.

A konfigurálás eredményeként, az IN bemeneti struktúra 2. és 3. bájtját átirányítottuk, hogy az ne a sebességet, hanem az aktuális pozíciót adja vissza. Így eltérően a 2.2. ábrán jelölttől, nem a sebességet kapjuk vissza, hanem a szögelfordulással arányos mennyiséget.

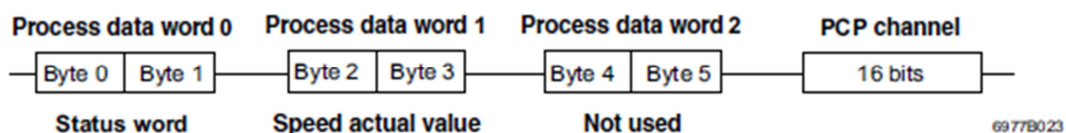
Alább található a folyamatábra.

Szervovezérlő inicializálása



2. 1. ábra

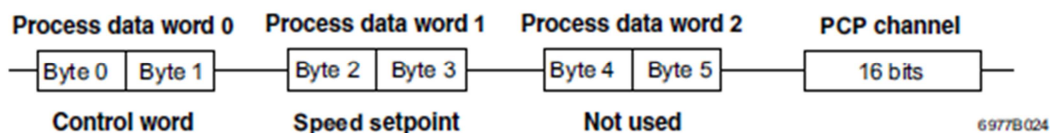
Servo vezérlő inicializálása



2. 2. ábra

IN Process word struktúrája

Az inicializálást követően a szervó motor sebességét egy DWORD-el tudjuk vezérelni, ugyanis alapértelmezés szerint az OUT Process data word 2. és 3. bájtja adja a sebesség szabályozási kör alapjelét (Set Point).



2. 3. ábra

OUT Process word struktúrája

Az ábrából kiderül, hogy a sebesség értékét 16 biten kell megadni, viszont 16 bittel a nagyobb helyiérték felé kell shiftelni 16-szor, ez megfelel 65536-tal való szorzásnak.

Egy szervó vezérlő modul be- és kimeneti változói:

Device	Process Data Item	I/Q	Data Type	Byte.Bit
# 2 IB IL EC AR 48/10A 0 . 2	INWORD0	I	WORD	0.0
# 2 IB IL EC AR 48/10A 0 . 2	INDWORD1	I	DWORD	2.0
# 2 IB IL EC AR 48/10A 0 . 2	OUTWORD0	Q	WORD	0.0
# 2 IB IL EC AR 48/10A 0 . 2	OUTDWORD1	Q	DWORD	2.0
# 2 IB IL EC AR 48/10A 0 . 2	~AI 48	I	OCTETSTRING[6]	0.0
# 2 IB IL EC AR 48/10A 0 . 2	~AO 48	Q	OCTETSTRING[6]	0.0

2. 4. ábra

Szervo vezérlő I/O változói

Az előbbi terminológia szerint a Process Word-ök 0. és 1. Bájta képez egy WORD-öt, míg a 2. és a 3. bájtt DWORD adattípust alkot a mögötte levő 2 általunk nem használt bájjal.

A sikeres inicializáláshoz szükséges a felparaméterezés után, hogy a szervo vezérlő modult az állapotgépe szerint a megfelelő állapotba juttassuk.

Ehhez az alábbi programrészletre van szükség:

if configured then

```
IF ((status and word#2#0000000001001111)= word#2#0000000001000000) THEN  
    control:=INT_TO_WORD(6);  
END_IF;
```

Ha már az előbbi szekvenciákat kiküldtük, a státusszavát megvizsgáljuk a szervo vezérlőnek:

Ha az inicializálás készen van, akkor a vezérlőmodul automatikusan „Start inhibit” állapotba lép, ez jelzi, hogy a hardveres és szoftveres inicializálás befejeződött. Ezt követően „Ready to operate” állapotba juttatjuk, amihez a 6 hexa értéket kell vezérlőszóként megadni.

```
IF ((status and word#2#0000000001111111)= word#2#0000000001000001) THEN  
    control:=INT_TO_WORD(7);  
END_IF;  
end_if;
```

Abban az esetben, ha „Ready to operate” állapotba lép a gép, akkor „ON” állapotba kell léptetni. Ehhez a 7 hexa értéket kell kiküldeni.

```
SpeedOUT:= Real_TO_DWORD(speed * 65536.0);
```

Ez a sor az előbbieken az „OUT szavak” fejezetben taglaltak szerint a kívánt Set Point beállítása érdekében 16 bittel a nagyobb helyiértékek felé kell eltolni a sebességértéket.

```

if speed <> 0.0 then
    IF ((status and word#2#0000000001111111)= word#2#000000000100111) THEN
        control:=INT_TO_WORD(15);
    END_IF;
end_if;

```

Ha a vezérlő programunknak átadott sebességérték 0-tól eltérő, és az állapotgép „ON” állapotban van, akkor az „Enable drive function” bekapcsolásával a motor hajtása engedélyezett állapotba kerül.

```

if speed = 0.0 then
    IF ((status and word#2#0000000001111111)= word#2#000000000100111) THEN
        control:=INT_TO_WORD(7);
    END_IF;

```

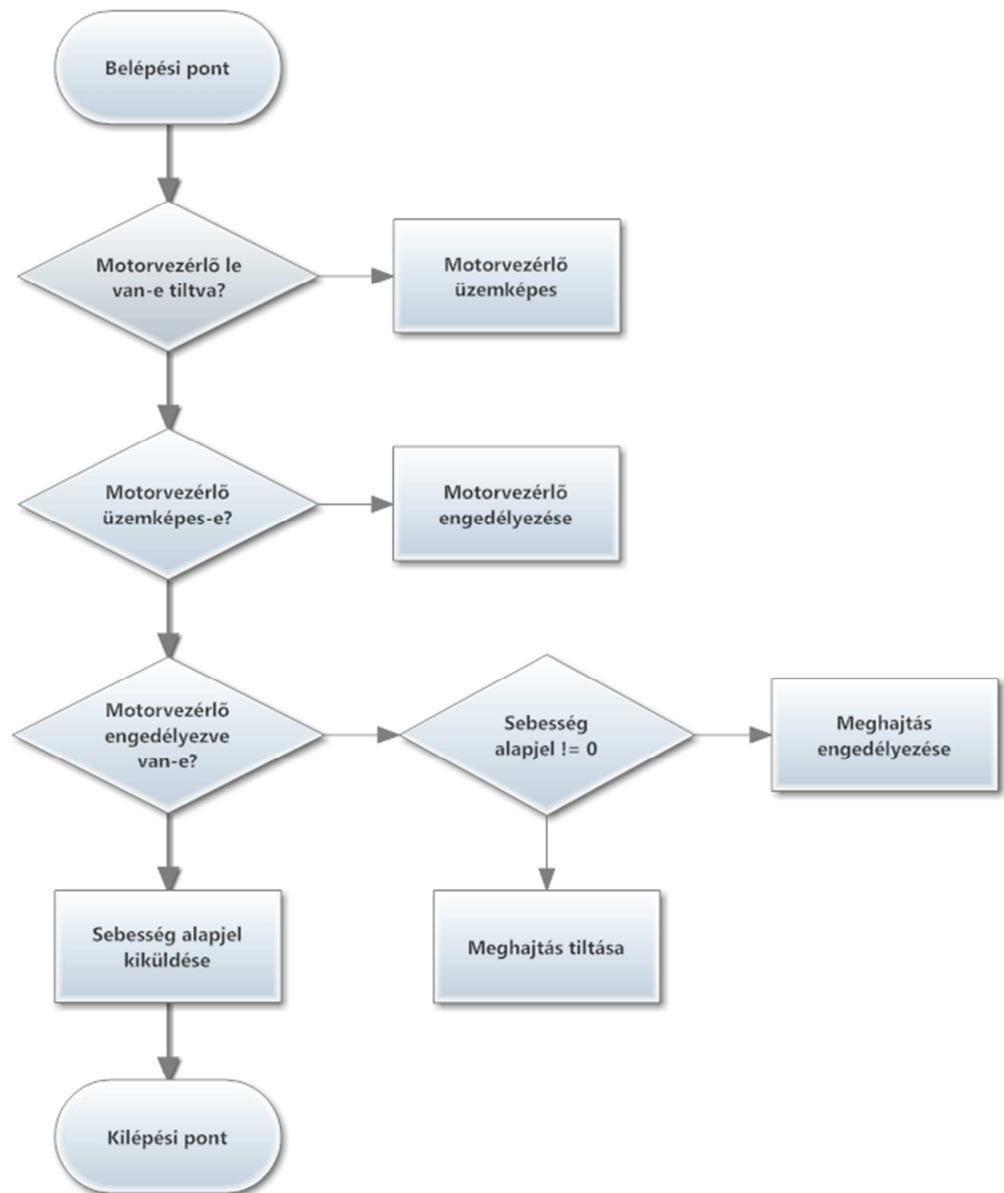
Ha sebességként 0-t adunk meg, és a motor hajtása engedélyezett állapotban van, akkor „ON” állapotba kell visszaléptetni, és a hajtás engedélyezést ki kell kapcsolni.

```

end_if;

```

Motorvezérlés



2. 5. ábra

Motorvezérlő gráf

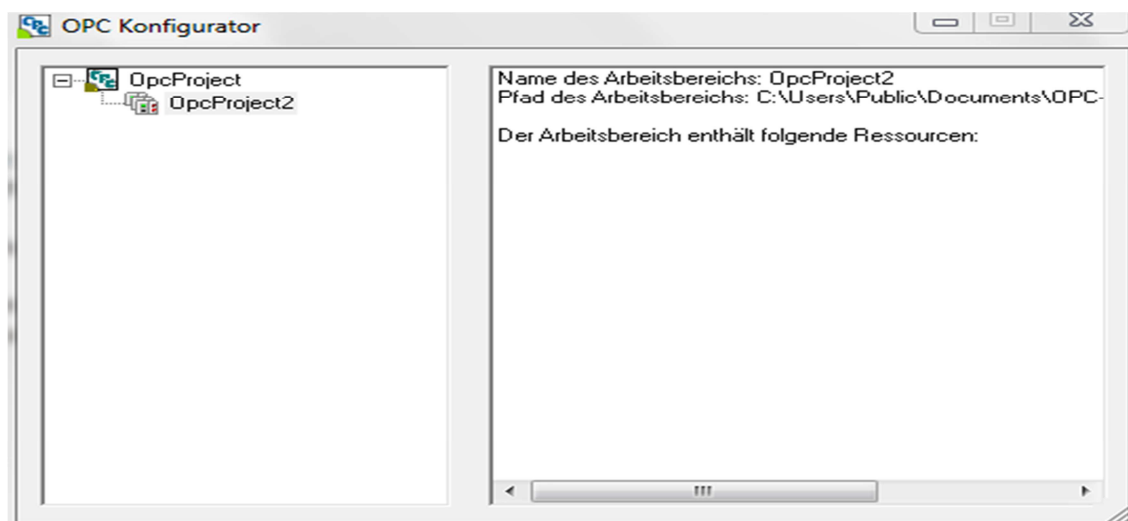
OPC szerver:

Az OPC (Open Platform Communication) kommunikációs platformot azért fejlesztették ki, hogy megvalósítható legyen egy valós idejű kapcsolat létrehozása egy PC és egy távoli folyamatirányító rendszer között. A protokoll létrejöttének valódi oka, hogy általánosságban jellemző minden PLC illetve folyamatirányító rendszer gyártó cégre, hogy az általuk használt kommunikációs protokollt, ami a PC és az eszköz közötti kommunikációt vezérli titkosítják.

Ezzel párhuzamosan viszont megjelenik az igény, hogy folyamatirányító rendszer folyamatváltozóinak valós idejű értéke számítógépes rendszerben (pl. SCADA szoftveren) megjeleníthetővé váljon. Ennek kielégítésére a hardver gyártók elkészítik saját OPC szerverüket és hozzá az OPC kliensüket is. A robot programozásakor egy Matlab program kapcsolódik a Phoenix Contact OPC szerveréhez, és ettől a szervertől kérdezi le a Matlab-os kliens az OPC-s változók értékét.

A 3D-s kamera képét Matlabon keresztül olvassuk ki, illetve Matlab segítségével dolgozzuk is fel, továbbá Matlab segítségével küldjük el a feldolgozás eredményét, a vezérlőjeleket a PLC-nek.

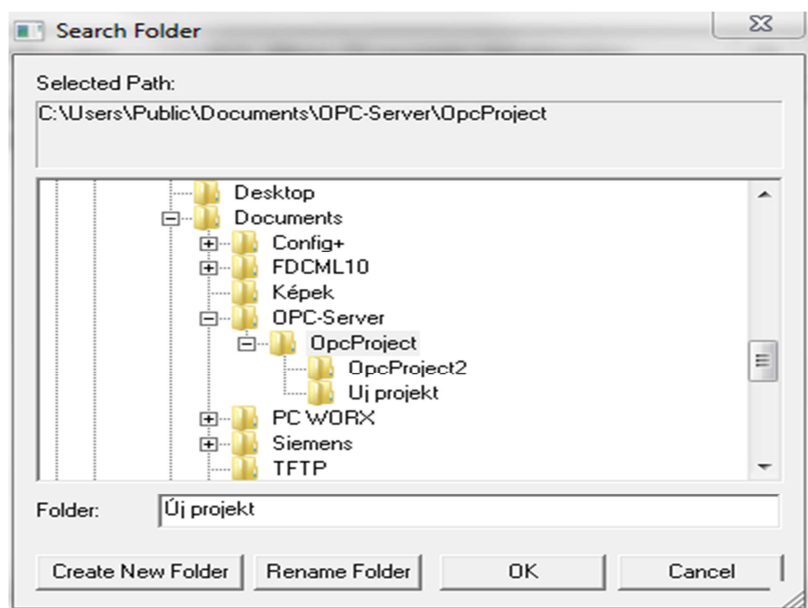
Először is létre kell hozni egy OPC szervert az ipari PC-n. Ehhez a Phoenix Contact AX OPC Server 2.30.01-es változatát használtuk. Az OPC konfigurátor megnyitása után az alábbi ablakot kapjuk:



3. 1. ábra

OPC konfigurátor kezdőfelülete

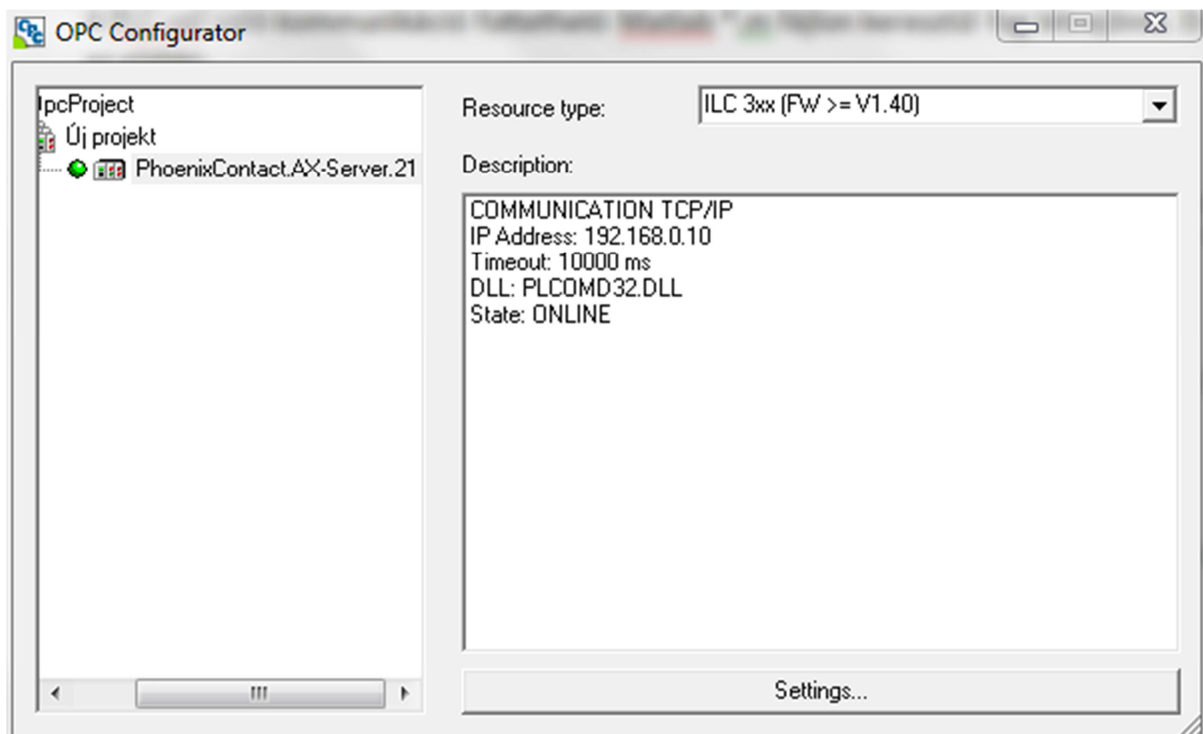
Az OpcProject ikonra bal egérgombbal kattintva létrehozhatunk egy új munkát (projektet)
New workspace menüpontra kattintva:



3. 2. ábra

Új projekt létrehozása

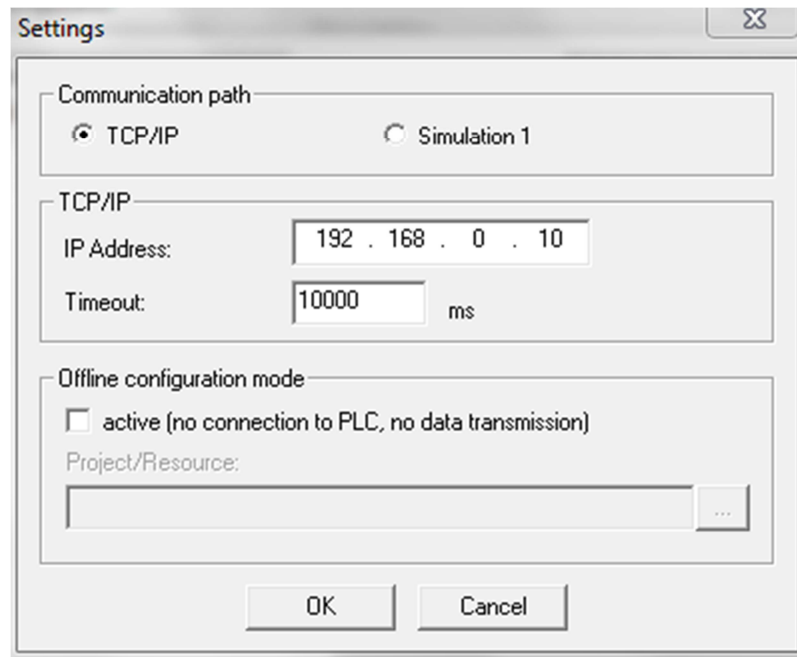
Ezt követően egy új erőforrást (resource) adunk hozzá a projekthez, majd beállítom a megfelelő PLC-t:



3. 3. ábra

Online kapcsolat

Már csak 1 lépés van hátra, az cél eszköz (PLC) IP címét kell beállítani. Ehhez a Settings... menüpontba kell belépni:



3. 4. ábra

IP cím és protokoll beállítása

A PLC-vel való kommunikáció futtatható Matlab *.m fájlon keresztül fog létrejönni. Ennek forráskódja az alábbi:

```
hostInfo = opcserverinfo('localhost');

%Ez a sor lekérdezi a helyi host állapotát. Visszatérési értéke a
%függvénynek egy struktúra mely tartalmazza az OPC szerver nevét

allServers = 'hostInfo.ServerID';

% a struktúra ServerID tulajdonsága tartalmazza az összes talált OPC
%szervet nevét

da = opcd('localhost', 'PhoenixContact.AX-Server.21');

% Létrehozunk egy OPC adatahozzáférési objektumot, melyen keresztül
le %tudjuk kérdezni az összes változó értékét
%localhost a már meglévő struktúránk, 'PhoenixContact.AX-Server.21'
névnek %pedig meg kell egyeznie a fentebb létrehozott OPC szerver
nevével

connect(da);
```

```

%Kapcsolódás a létrehozott OPC hozzáférési ponthoz

group1 = addgroup(da) ;

%Kapcsolódás után létrehozunk egy csoportot, aminek az alapja az OPC
%hozzáférési pont

itm1 = additem(group1, ' PhoenixContact.AX-
Server.21.Robot1.elolkamera');
itm2 = additem(group1, ' PhoenixContact.AX-
Server.21.Robot1.balrakamera');
itm3 = additem(group1, ' PhoenixContact.AX-
Server.21.Robot1.jobbrakamera');
itm4 = additem(group1, ' PhoenixContact.AX-
Server.21.Robot1.manage');

%Ezt követően a létrehozott OPC csoportunkhoz hozzáadjuk azokat a
változókat, amelyeket írni szeretnénk

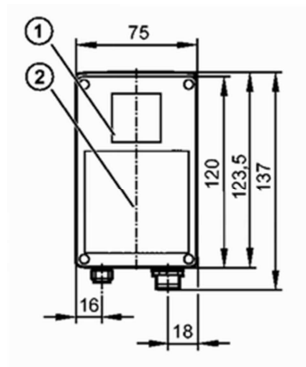
disp('OPC init kész');

%Kiírja a főablakba, hogy 'OPC init kész'.

```

IFM O3D200 PMD kamera

A robot megfelelő navigálásához, nem csak síkbeli lézerszenner alapján történő akadály és céltárgy felismerés szükséges, hanem arra is szükség lehet, hogy a robot a kerekeinél magasabban, saját testmagasságában található akadályokat is felismerje.



1.: CMOS PMD szenzor

2.: Megvilágító rész

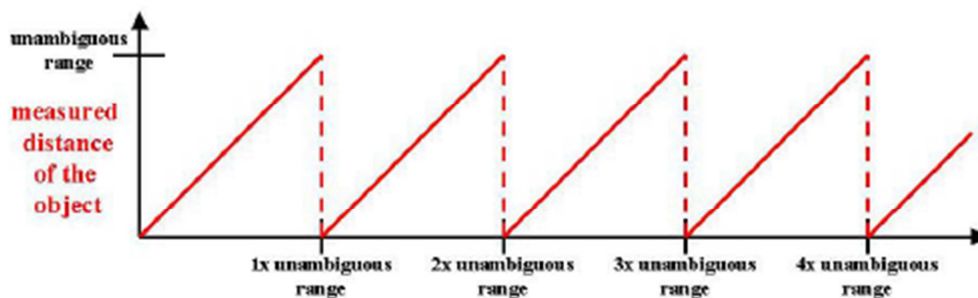
4. 1. ábra

PMD kamera felépítése

A probléma megoldásában az Ifm Gmbh O3D200 típusú PMD kamerája volt a segítségemre. Ez a kamera egy ipari kivitelű (IP 65-ös védettségű) távolságmérő eszköz, mely 64X50 db pixelen képes rögzíteni a látószögében lévő objektumokat. A kamera nagy integráltsági fokú CMOS szenzort tartalmaz. A szenzorral megoldható, hogy úgy készítsen a kamera valós idejű 3D-s képet, hogy ahhoz különlegesen nagy plusz számítási kapacitás ne legyen szükséges.

A működési elve a kamerának azon alapszik, hogy a kamerába épített adó modulált infravörös fénnel (LED fényforrásból) borítja be a látószögébe eső teret, és az onnan visszaverődő hullámok a megfelelő CMOS pixelbe jutnak vissza. Ezen módszerrel megvalósítható, hogy szürkeárnyaltos és a távolságot is leíró képhez jussunk. A teljes kép ezen CMOS pixelek által alkotott mátrixból (ez a PMD mátrix) tevődik össze. A kamera egy ilyen képet körülbelül 30 ms alatt dolgoz fel.

Az adott pixelhez rendelt távolságérték futásidőn (Time of Flight) alapuló mérésre van visszavezetve, ugyanis az időt pontosan lehet mérni, továbbá a fotonok sebessége ismert az adott közegre vonatkozóan.



4. 2. ábra

Mérési eredmény alakulása

A kamera képfeldolgozó egysége egy jól meghatározott (unambiguous) mérési tartománnyal rendelkezik, ez 6.5 méter. A fenti ábrából jól következik, hogy a szoftverünk által hozzáférhető távolság adat 6.5 méter modulója. Így például egy tárgy, ami a valós térben kb. 10 méter távolságban helyezkedik el a kamerától, a mérési adatban 3.5 méteres távolság jelenik meg.

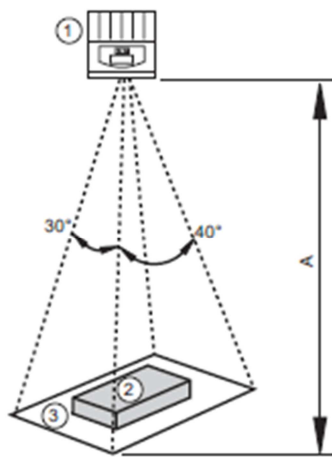
Mivel a képalkotó CMOS szenzor a visszavert hullámokat érzékeli az infra tartományban (a foton forrás is azért ebben a tartományban sugároz, mert a szilícium alapú félvezető szenzorok ebben a hullámhossztartományban a legérzékenyebbek), fontos tényező a reflektáló felület anyagi minősége, a hullámok beérkezési szöge, illetve a környezetben található egyéb fényforrások jelenléte is.

Legpontosabb mérés leginkább laborkörülmények között valósítható meg. Minden fényforrás, mely az infra tartományban is sugároz zajforrásként értelmezhető, mely jelentősen csökkenti a mérés pontosságát.

Komoly kompenzálhatatlan problémát jelent a több utas jelterjedés. Ennek gyakorlati példája egy meglepő mérési eredmény volt: a laborban található gömb alakú tartállyal rendelkező kompresszor egy, a kamerával merőlegesen elhelyezkedő szerelőszeletrény előtt volt elhelyezve. A készített képen viszont a kompresszor ívesebb felülete nagyobb átlagtávolság értékkel volt megjelölve, így valószínű, hogy a szenzor a plafonról visszavert jelet érzékelt,

amely hosszabb szakaszon ért vissza a kamera szenzorjába, mint a kompresszor valós távolsága. Természetesen a merőleges felületű fém szerelőszekrény távolsága pontosan jelent meg a képen.

A mérési tartomány a mérés elvéből adódóan változhat. A megvilágító fényforrás (IR LED) fényének intenzitása a távolság négyzetével csökken, így detektáláskor ez is problémát okozhat. Egy sötét test reflexitása csupán 5 % is lehet, így a szenzorba nagyon kis fényintenzitás jut. Ebből és az előzőből következik, hogy sötét testek távolságmérése zavaró tényezőkkel terhelt, így a specifikáció szerint csak 2-4 méteres távolságig képes a kamera a sötét színű objektumokat mérni. A világos felületű, kis emissziós tényezőjű tárgyakat akár 150 méteres távolságban is képes érzékelni a kamera, ami sokszorosan túllépi a 6.5 méteres alap mérési tartományt.



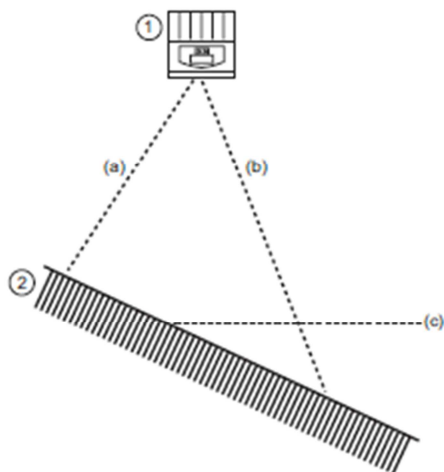
A kamera, mivel függőlegesen és merőlegesen nem ugyanannyi pixelről készít képet, függőlegesen 40 fokos, vízszintesen 30 fokos kúpszögben készít felvételt.

4. 3. ábra

PMD kamera látóképe

További mérés technikai problémát jelent a kamera véges felbontóképessége. Ebből származó hiba lehet, hogy nagyobb távolságban egy tárgy egyre nagyobb felületegysége tartozik egy pixelhez, így a tárgy felületének egyedisége, illetve a felület elhelyezkedése problémát okoz.

Az alábbi ábra jó hasonlata a lehetséges problémának:



4. 4. ábra
Érzékelési probléma

A specifikáció szerint 1 méteres távolságból a kamera egy 12X12 mm-es felületet képes 1 pixelként leírni, 2 méteres távolságból pedig 25X25 mm-es felületegységet fog 1 pixelként megjeleníteni a kamera. Ebből kifolyólag, hogyha az ábrán látható elrendezés megvalósul, akkor a pixelhez tartozó felületegységről különböző időközökkel verődnek vissza hullámok („a” és „b” jelű). Ilyen esetben a „c” jelű átlagos távolságot fogja a kamera a pixelhez tartozó távolságértékként megjelölni. Ebből következik, hogy legalább kétszeres felületegység szükséges ahhoz, hogy a tárgy megfelelő

távolságban látszódjék a kamera képén.

Ismételten problémát okozhat, ha a kamera/tárgy elmozdul exponálás közben. A kamera egy távolságkép elkészítéséhez 4 darab képet készít, egymás után gyorsan exponálva, és problémát okoz az, hogyha pixelhatáron történik az elmozdulás, tehát a tárgy egy része átugrik egy másik pixelre, a 4 mérés valamelyike közben. Ennek elkerülése végett a kamera és a tárgy rögzített elhelyezése szükséges, viszont ez rögtön lekorlátozná az eszköz használhatóságát. Tipikus ipari alkalmazás lehet például a kamera fix pozíciója mellett a fotószalagon haladó tárgyak vizsgálata, de ebben az esetben sem küszöbölhető ki a fent említett hibajelenség.

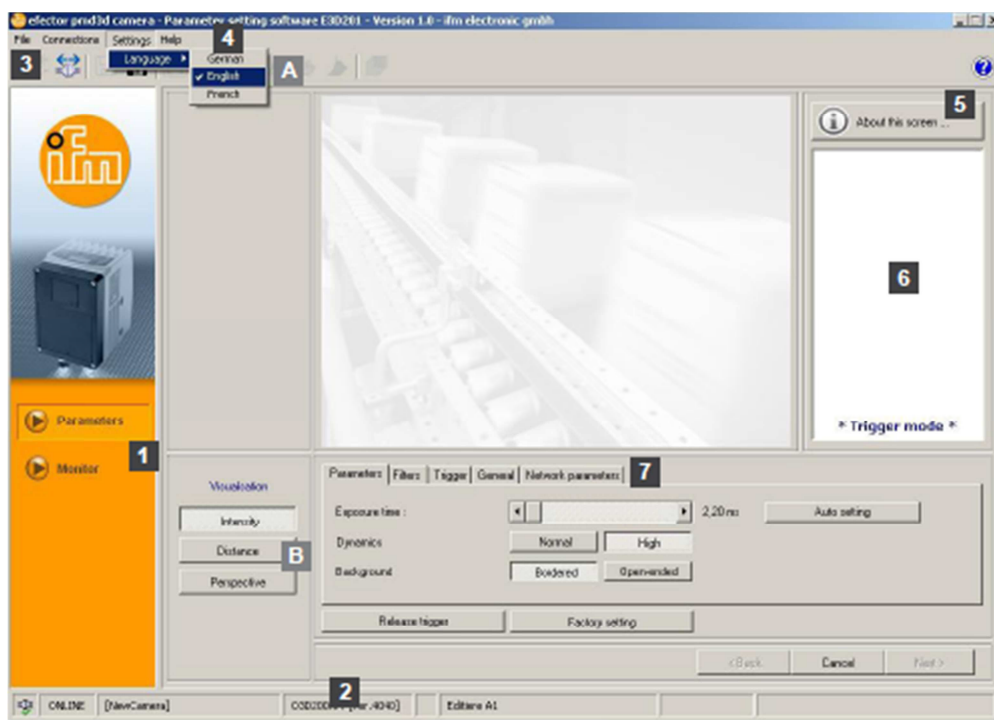
A specifikációban említett egyéb speciális hibalehetőség az, hogyha több kamerát egymáshoz közel (kamera látótávolságon belül) helyezünk el, és esetlegesen azokat egyszerre exponáltatnánk. Ennek elkerülése érdekében az első kamera „ready” kimenetét kell a következő „trigger” bemenetére kötni, és így tovább létrehozunk egy láncot, ahol csak az első kamera ütemezésével kell foglalkoznunk.

Fontos továbbá megjegyezni, hogy a kamera infra megvilágítójának sugara, egyéb infra jeleket használó eszközöket (opto szenzorokat) befolyásolhat, így fontos átgondolni a telepítését, illetve a többi szenzort/eszközt nem optikai működési elvűre (pl. induktív, kapacitív) választani.

Gyári konfiguráló szoftver

Miután kézhez kaptuk a kamerát először is a csatlakozó interfészeket kellett megkeresni. A csatlakozás 2 db M12-es kábellel volt megvalósítva. A 8 pólusú kivitel a 24 V-os egyenáramú tápellátásért felel, míg a 4 pólusú az Ethernet interfészhez való kapcsolódást biztosítja.

Ahhoz, hogy kipróbáljam a kamera működőképességét a gyári „Efector PMD3D camera” szoftvert használtam.



5. 1. ábra

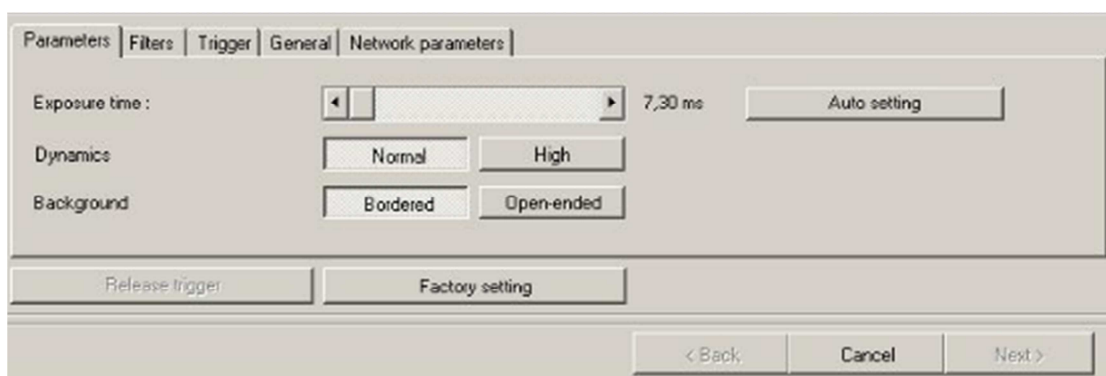
Gyári konfiguráló szoftver

Az alap bejelentkező ablak 2 fő ágat különböztet meg: paraméterezés és monitorozás. A kamera kipróbálásához a monitorozás megfelelő, viszont minekünk a legfontosabb a paraméterezés menüpont alatt található. A dokumentációban említett, illetve általunk tapasztalt tény, hogy az expozíciós idő hatással van a mérési tartományra. A specifikáció szerint, ha egy tárgyról nagyon kicsi a visszareflektált hullám, akkor a kamera a legnagyobb távolságot (6.5 métert) fogja hozzárendelni. Ezt hívják alulexponált felvételnek.

Másik probléma a túlexponáltság. A mi esetünkben ez okozott problémát. A tapasztalataim során úgy következtettem, hogy a kamera jelfeldolgozása integráló jellegű, tehát az

exponálási idő alatt egy reflektáló felületről (holdról), vagy egy önálló IR forrással rendelkező objektumból származó fényáram összegét méri, és ebből következtet az objektum távolságára.

A robot testmagasságában felmerülő akadály felismerése olyan feladat, ahol nem a nagy távolságokra történő mérésre van szükség, hanem a kicsire. A kamera valószínűleg fix minimum és maximum intenzitásértékekre van kalibrálva, így nem mindegy, hogy mennyi ideig hagyjuk a kamera szenzorját integrálni (exponálni). 20 ms-os exponálási időt választva a kamera kb. fél métertől volt használható, tehát a viszonylag nagy exponálási idő ilyen távolságot követelt, hogy a reflektált jel integrálja ne haladja meg a maximum értéket. Így, ha kb. 20 centiméteres távolságtól szeretnénk használni a kamerát, akkor nagyon le kell csökkenteni az exponálási időt, hogy az reflexióintenzitás összege alacsony maradjon. Ehhez, ahogy az ábrán is látható a 7.3 ms-os exponálási idő volt a legmegfelelőbb számunkra.



5. 2. ábra

Exponálási idő beállítása

$$\frac{1}{Távolság} = c * \int_{t_0}^{t_0 + T_{exposure}} I(t) * dt$$

Tehát a mért távolság fordítottan arányos a reflexióintenzitás idő szerinti határozott integráljával.

O3D201 kamera képfeldolgozása:

A lézerszkennerek kialakításából adódóan csak egy síkban, kis kúpszöggel képes „képet” alkotni a környezetéről, így a magasabban elhelyezkedő akadályokat egy másik érzékelővel kell vizsgálni. Ezt a célt szolgálja a PMD kamera.

A kamerával a kapcsolatot Matlabbal építjük fel, de előtte a kamerát fel kell paraméterezni a saját gyári szoftverével.

A kamerával való kapcsolat felépítéséhez az O3D2xxCamera.dll fájlt használjuk illetve O3D2xxCameraMatlab.h headert, mely tartalmazza azokat a függvényeket, amelyekkel tudjuk vezérelni a kamerát:

Forráskód:

```
dllname = 'O3D2xxCamera'; headername = 'O3D2xxCameraMatlab.h';

%Megadjuk a kamerához szükséges dll és header fájl nevét változók
%formájában.

IP = '192.168.0.69'; XMLPort      = uint32( 8080 );

%IP változónak megadjuk a kamera gyári IP címét, illetve XMLPort
%változóban megadjuk a gyári portcímét

hwnd = libpointer( 'uint32Ptr', 0);  FWversion = libpointer(
'stringPtr', 'xxxxxxx' );

SensorType = libpointer( 'stringPtr', 'xxxxxxx' );

%létrehozunk pointerket, melyek a dll kezeléshez szükségesek

loadlibrary O3D2xxCamera O3D2xxCameraMatlab.h;

%könyvtárfájl megnyitása

if ~libisloaded( dllname ) [notfound, warnings]=loadlibrary(
dllname, headername ); end;% libisloaded jól működik

for c = 1:3
    [ answer, hwnd, IP, FWversion, SensorType ] = calllib(
'O3D2xxCamera', 'O3D2XXConnect', hwnd, IP, XMLPort, FWversion,
SensorType );
    if answer == 0 disp( 'Camera Connected' );
        calllib( 'O3D2xxCamera', 'O3D2XXPerformHeartBeat' ); break;
    else calllib( 'O3D2xxCamera', 'O3D2XXDisconnect', 1 );
        disp( strcat( 'Failed to connect, attempt no', num2str(c) )
);
```

```

        if ( c == 3 ) error( 'Failed to connect' ); end; end; end;

%Az alábbi kód felépíti a kapcsolatot a kamerával, három sikertelen
%kapcsolódási kísérlet után kilép, és hibaüzenetet ír ki.

calllib( 'O3D2xxCamera', 'O3D2XXSetTrigger', 4 );

%triggerelési mód beállítása, a beállított mód a szoftveres
%triggerelés

head = libpointer('int32Ptr', zeros(1,21*4)); SIZE =
uint32(64*50*4);
dat = libpointer('uint8Ptr', zeros(1,SIZE)); TCPport = int32(50002);
cam = struct( 'dllname', dllname, 'dat', dat, 'SIZE', SIZE,...
'head', head, 'TCPport', TCPport);

%setfrontend = libpointer( 'int32Ptr', [1, 0, 1,2000, 2000, 0] );
%calllib( dllname, 'O3D2XXSetFrontendData', setfrontend );

disp('Kamera init kész');

%A kamera inicializálása készen van.

```

Az alábbi kódrészlet végén látható 2 kikommentezett sor. Ha nem kommenteztük volna ki, akkor a kódunk átírta volna a kamera paramétereit, ezzel a mérési tartományt kitolta volna 6.5 méter és fél méter közé, ami lehetetlenné tette volna a közeli tárgyak érzékelését. Ennek oka, hogy az exponálási idő alapesetben nagyobb, mint 7.3 ms.

SICK LMS100 lézerszkennerek bemutatása

Mint korábban említettem róla a robot saját testmagasságában egy 3D-s lézerkamera segít információt gyűjteni, viszont az az információ nem elég pontos, nem eléggé nagy a felbontása, illetve nem is annyira megbízható, hogy az alapján a robot önmagát navigálni tudná. Ennek okán a robot konstruktőrei egy nagy látótávolságú, széles látószögű, illetve pontos mérési eredményt szolgáltató lézerszkennert telepítettek a robot gördülési magasságának tartományában. Ez a lézerszkennerek egy német gyártmányú SICK LMS100-as lézerszkennerek, mely IP 65-ös védettségi fokkal rendelkezik.

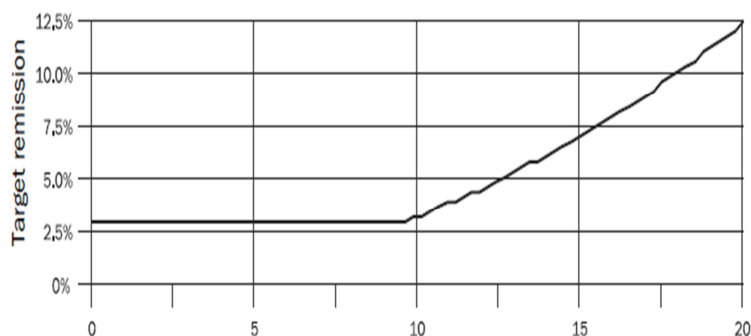
Írányítás szempontjából a legfontosabb jellemzője a széles látószög, ezen belül a nagy felbontás és a megfelelően nagy látótávolság nagy felület pontos leképezését teszi lehetővé. A szkennerek látószöge 270 fok, amit a mi esetünkben fejjel lefelé kell érteni, ugyanis a lézerszkennerek felfogatása így volt a legmegfelelőbb. Ennek eredményeképp tengelyesen tükrözött mérési felületet kapunk a mérés eredményeképp. A lézerszkennerek ezt a tartományt 0.5 és 0.25 fokos felbontással képes végigpásztázni. A pásztázás szögtartományonbeli enkóderrel történik, tehát egyszerre csak egy pozícióban történik mérés, ez zavarvédelem szempontjából jó, hiszen ha több IR forrás egyidőben pásztázná a területet akkor a hullámok többutas jelterjedése csökkentené a mérés pontosságát. Attól függően, hogy a lézerszkennerek felbontása 0.25 fok/mérés, vagy 0.5 fok/mérés a mérési frekvencia változik, több mérési ciklushoz jutunk, hogyha a fél fokos felbontást választjuk. (Így 50 Hz-es maximum frekvenciát tudjuk kihasználni).

A mérés elve a futásidő mérésén alapszik. Mindegyik pozícióhoz kapcsolódóan a szkennerek egy IR led forráson keresztül foton impulzust ad ki, és méri a belső processzoros egység a visszavert impulzus megjelenési késleltetését és intenzitását. Így az eltelt idő mérésére van visszavezetve, amit pontosan lehet mérni. Minden mérési pont után az enkóderes léptető 0.5 vagy 0.25 fokkal léptet, és az előbbieket nyomán méri le a teljes 270 fokos szögtartományt.

Erre a 270 fokos tartományra vonatkoztatott mérésre kell érteni az 50 Hz-es mérési frekvenciát.

Fontos paraméter a mérési távolságtartomány. A mérés fizikai elvéből adódóan a detektor

Scanning range LMS100/LMS111/LMS12x/LMS13x/LMS173/LMS182
and LMC12x/LMC13x



7. 1. ábra

Emisszióképesség és látótávolság függvénye

csak megfelelő jelintenzitású reflektált hullámot képes érzékelni, így fontos a reflektált jelerősség egy bizonyos szint fölött való tartása. Ebből kifolyólag a katalógusban szereplő 20 méteres mérési tartomány csak a mérendő felületek 13 %-os reflektáló képessége fölött garantált. Általánosságban igaz, hogy a környezeti zajok miatt a nagyobb reflexiós képesség előny a pontosság

szempontjából. Ebből adódóan szűkül a lézerszkennerek alkalmazhatósági lehetősége, ugyanis a legjobb környezet a világos színű IR tartományban jól reflektáló felületeket tartalmazó tér. Ilyen környezet előfordul például a Honvéd Kórházban található alagsori helyiségekben, ahol az önműködő szállítórobotok navigációjához megfelelő a lézerszkennerek. Egy másik alkalmazás lehet a rosszul reflektáló közegben történő jól reflektáló felületű tárgy megkeresése, követése. Ilyen feladat a szakdolgozat végén tárgyalt tárgykövetési feladat.

Ahhoz, hogy a mérési eredményekhez hozzáférjünk a lézerszkennerek kommunikációs interfészét kell használnunk.

Ezek lehetnek Ethernet, CAN illetve RS-232-es soros vonali interfész. Mivel a mobil robot egy kis LAN hálózatot is tartalmaz, mely egy Phoenix Contact switch-el lett kialakítva, így logikus döntés az Ethernetes interfészt használni a kommunikációhoz. A kommunikáció lebonyolításához Matlab programot használtam, mely a Matlab saját TCP/IP stackjét használta. A felparaméterezés történhet a SOPAS nevű gyári szoftveres felületről, vagy saját Matlabos telegramokon keresztül. Előbbi esetben a mérési eredmények feldolgozását követően a lézerszkennerek 3 darab diszkrét BOOL kimeneti változóját használhatjuk a robot navigálásához.

Gyári kalibráló szoftver (SOPAS) használata:

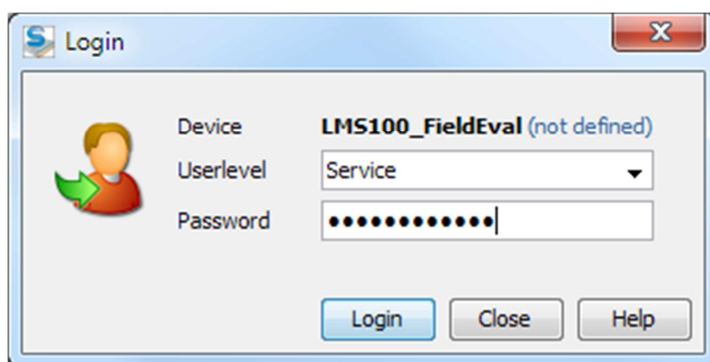
Nem minden esetben van idő és lehetőség saját konfiguráló szoftver írására (a telegram lista alapján), így a lézerszkennerral való gyakorlati megismerkedést a SICK gyári SOPAS Engineering Tools nevű szoftverrel kezdtük.

Ahhoz, hogy a szoftverrel bármilyen érdemi módosítást végre tudjunk hajtani a lézerszkennerek ROM-jába letöltött firmware-en a jogosultsági szinteket be kell állítani.

Ezek:

- Maintenance
- Authorized Client
- Service

Ezen jogosultsági szintek mindegyikéhez különböző ROM hozzáférési jogok vannak



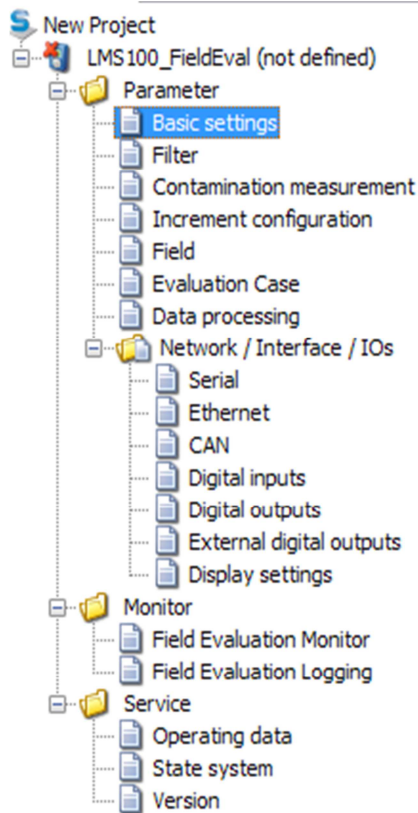
rendelve, a lézerszkennerekre rátöltött firmware bizonyos részeit lehet csak megváltoztatni. A hozzáférési jogok lefelé haladva nőnek. A teljes konfiguráláshoz a Service szint a legmegfelelőbb, ehhez a jelszó „servicelevel”.

7. 2. ábra

Jogosultsági szint beállítása

A szoftver bemutatása során végig fogok tételesen haladni azokon a konfigurációs szinteken, melyek szükségesek a lézerszkennerek konfigurálásához:

A bejelentkező ablak baloldalán található a konfigurációs menüsáv. A „Basic settings” menüpont alatt kiválaszthatjuk, hogy a 270 fokos szögterület végigszkennele hány foktól kezdődjön, illetve hány fokig terjedjen. Alap beállítás a -45 – 225 fokos tartomány kiválasztása. További számunkra fontos beállítás a szkennelési frekvencia, ami jelen esetben 50 Hz-re lett beállítva.



7. 3. ábra

Paraméterfa

A „Filter” fül alatt beállítható, hogy kód (pl. porfelhő) által okozott tükröződés ne okozzon akkora mérési bizonytalanságot az esetleges reflexió miatt. A bekapcsolása esetén a lézerszkennerek elülső oldalán egy érzéketlenségi zóna jön létre, így abban a tartományban a szkennerek nem fog érzékelni reflexiót. Ide tartozó beállítás az „N-pulse to 1-pulse filter”, mely olyan algoritmust tartalmaz, ami a legelső reflektált hullámot (pl. ködfátyolból adódót) nem veszi figyelembe a mérés szempontjából. Beállítható részecske szűrő is, ami a porszemcsék, hópelyhek, vagy esőcseppek esetén segít az interferencia elkerülésében, illetve belső átlagképzés is beállítható, ami kisfrekvenciás szűrőként is felfogható. Ez utóbbi két szűrő, által manipulált mérési eredmény az Ethernetes interfészen nem

jelenik meg, csak a később tárgyalt területfigyelés esetén befolyásolja a mérést.

A „Contamination measurement” abból a szempontból érdekes, hogy a lézerszkennerek képesek detektálni a saját, érzékelőjét védő plexi szennyeződését. Megadhatjuk, hogy milyen reflexió intenzitásnál kapjuk figyelmeztetést, vagy a hiba jelzést.

Az „Increment configuration” beállításnál megadható, hogy a szkennerek mozgása során, milyen visszacsatoló enkóder jelből nyerje a lézerszkennerek az információt. Ennek a beállításnak akkor van értelme, hogyha a szkennert egy elforgatható tengelyre szerelik rá, és azt egy motorral hajtják aminek a tengelyére egy enkóder tárcsa van szerelve. Ilyenkor a statikus mezőbeállítás nem lenne használható. Megadható, hogy egyfázisú, vagy kétfázisú irányérzékeny kvadratúra enkóderrel csatlakoztatunk a szkennerekhez.

A „Field” menüpont számunkra a legfontosabb, ugyanis itt tudjuk definiálni a lézerszkennerek érzékenységi mezőit. A robot navigálása szempontjából fontos, hogy érzékelni tudja az előtérben elhelyezkedő akadályokat, illetve akadályok felmerülése esetén tudja a robot, hogy tőle jobbra, illetve balra található-e akadály. Ez az a paraméter, melynek állításához feltétlenül szükséges a legmagasabb jogosultsági szinttel való belépés.

SICK lézerszkennerek kezelése

A fentebb említettek szerint a lézerszkennerek a felkonfigurálása után 3 BOOL kimeneten jelzi az egyes Fieldekre vonatkozóan, hogy mely területen található akadály.

Ennek a 3 bitnek a hatását egy „sick” nevű létrában megírt program kezeli le, az alábbi forrás segítségével:



8. 1. ábra

I/O átadása logikai biteknek

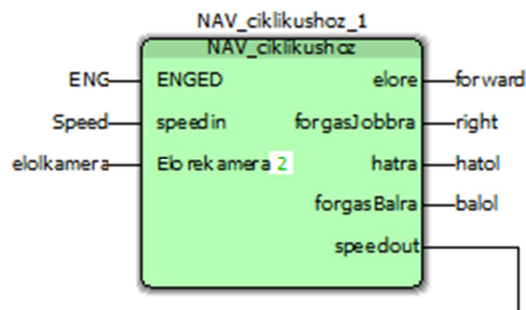
A programrész nagyon egyszerű, a lézerszkennerek kimeneteihez rendelt változókat átadja más változóknak, amelyek a robot vezérléséhez szükségesek.

A PLC-n futó vezérlőprogram

A vezérlőprogram tulajdonképpen 2 részre bontható fel. Az első rész a logikai vezérlést végzi, a második a fizikait. A logikai jelek vezérlik azt a programrészt, mely a szervomotorok fizikai vezérléséhez szükségesek. A szervomotorok vezérlésének menetét feljebb már tárgyaltam.

Logikai vezérlés:

A logikai vezérlésért egy NAV_ciklushoz nevezetű programrész felelős. Bemeneti paramétere egy statikus engedélyező jel (ENGED), melynek megszakításával a robot leállítható.



9. 1. ábra

Logikai vezérlő meghívása

Az alábbi Function Block vezérlőjelei egy automata üzemmódban értelmezhető vezérlőjelek, melyek akkor kerülnek érvényre, ha nem akarjuk manuálisan vezérelni a robotot, vagy nem következett be valamilyen speciális esemény, melyekre a későbbiekben fogunk kitérni.

A forráskód az alábbi:

```
IF ENGED=TRUE THEN
```

```
  if not manover then
```

```
  if not Elorekamera then
```

```
  IF Elore_Bit=TRUE THEN (*lezárva*)
```

```
    IF BalAkadaly=TRUE THEN (*lezárva*)
```

```
      IF JobbAkadaly=FALSE THEN(*ha elől és baloldalon
```

```
ak. van*)
```

```
      előre:=FALSE;
```

```

        ForgasJobbra:=TRUE;
        ForgasBalra:=FALSE;
        hatra:=FALSE;
    else    (*ha mindenhol akadály van, menj hátra*)
        elore:=FALSE;
        ForgasJobbra:=FALSE;
        ForgasBalra:=FALSE;
        hatra:=FALSE;
        manover:=1; (*manőver indítása*)
    end_if;
else
        elore:=FALSE;
        ForgasJobbra:=FALSE;
        ForgasBalra:=TRUE;
        hatra:=FALSE;
    end_if;
If    not Elore_bit and BalAkadaly and not JobbAkadaly then (*lezárva*)
        elore:=FALSE;
        ForgasJobbra:=TRUE;
        ForgasBalra:=FALSE;
        hatra:=FALSE;
    end_if;

If    not Elore_bit and JobbAkadaly and not BalAkadaly then (*lezárva*)
        elore:=FALSE;
        ForgasJobbra:=False;
        ForgasBalra:=TRUE;
        hatra:=FALSE;
    end_if;
end_if;

```

Abban az esetben, hogyha az Elore_bit 1-es, tehát a lézerszkenner visszajelzi, hogy az előtte lévő térrészben van akadály, akkor a program megnézi, hogy tőle jobbra, illetve balra levő területen jelez-e a lézerszkenner, és a megfelelő logikai parancsot adja ki a fizikai vezérlő blokknak az „elore”, „ForgasJobbra”, „ForgasBalra” és a „hatra” nevű változókon keresztül.

```

IF Elore_Bit=FALSE THEN (*lezárva*)
    elore:=TRUE;
    hatra:=FALSE;
    ForgasJobbra:=FALSE;
    ForgasBalra:=FALSE;
end_if;
if Elore_Bit and JobbAkadaly and Balakadaly then (*lezárva*)
    elore:=FALSE;
    hatra:=TRUE;
    ForgasJobbra:=FALSE;
    ForgasBalra:=FALSE;

```

```

        end_if;
    else
        elore:=FALSE;
        hatra:=TRUE;
        ForgasJobbra:=FALSE;
        ForgasBalra:=FALSE;
    end_if;
    else
        elore:=FALSE;
        hatra:=FALSE;
        ForgasJobbra:=FALSE;
        ForgasBalra:=FALSE;
        speedout:=0.0;
    END_IF;
    speedout:=speedin;

```

Ennek a sornak a feladata az általunk adott érték, a „speedin” (mely arányos a sebességgel) átadása a fizikai vezérlő programrésznek.

```

else
    speedout:=0.0;

```

Ha az engedélyező bemenet nem aktív, akkor a fizikai vezérlő 0-s sebességértéket kap, így megáll a robot.

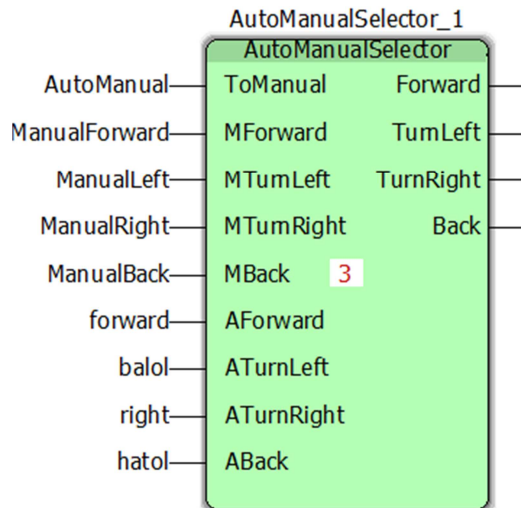
```

end_if;

```

A logikai vezérlés másik feladata, a vezérlőjelek multiplexálása a fizikai vezérlő rész felé. Erre azért van szükség, mert előfordulhat, hogy mi kézzel, manuálisan szeretnénk vezérelni a robot mozgását, illetve előfordulhat olyan eset is, amikor felül kell bírálni a lézerszkennertől kapott jeleket, és a robotot valamilyen határciklusból el kell mozdítani.

Erre a multiplexálásra szolgál az auto/manuál átváltó funkcióblokk, melynek „AutoManualSelector” a neve.



9. 2. ábra

Multiplexer

Forráskódja:

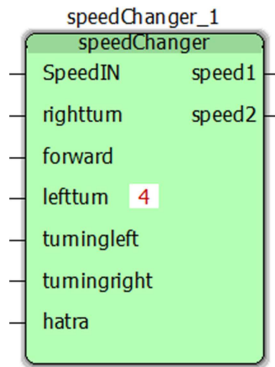
```
IF ToManual = True then
Forward:=MForward;
TurnLeft:=MTurnLeft;
TurnRight:=MTurnRight;
Back:=MBack;
```

```
else
Forward:=AForward;
TurnLeft:=ATurnLeft;
TurnRight:=ATurnRight;
Back:=ABack;
```

```
end_if;
```

Működésének lényege, hogyha a „ToManual” bemenetén logikai „1”-est kap, akkor felülbírálja a NAV_ciklushoz által a lézerszenzor alapján kiadott vezérlőjeleket és a következő logikai vezérlő fokozatnak a manuális vezérlő jeleket adja át, ellenkező esetben viszont átengedi az automatikus vezérlés hatását a következő szintnek.

Az utolsó logikai vezérlő szint a diszkrét logikai vezérlőjelet átalakítása folytonos vezérlőjelekké, melyért a „SpeedChanger” névvel ellátott funkció blokk a felelős.



9. 3. ábra

Sebességértékek meghatározása

A blokk működésének lényege, hogy az összes bemeneti kombinációhoz hozzárendeli a megfelelő sebesség értékeket a szervomotor számára:

```
if rightturn and not forward and not leftturn then
    if speedIN <> 0.0 then
        speed1:=SpeedIN -0.96*SpeedIN;
        speed2:= speedIN;
    end_if;
end_IF;
```

Ha a bemeneti vezérlés jobb oldalra terelné a robotot, akkor a jobboldali motor sebességét lecsökkentjük, a bal oldaliét, pedig meg hagyjuk eredeti értékén.

```
if forward and not leftturn and not rightturn then
    speed1:=speedIN;
    speed2:= Speed1;
end_if;
```

Ha előre kell menni, akkor mindkét szervomotor fordulatszáma megegyezik.

```
if leftturn and not forward and not rightturn (*and not turning*) then
    if speedIN <> 0.0 then
        speed2:=SpeedIN -0.96*SpeedIN;
        speed1:= speedIN;
    end_if;
end_IF;
```

Ha balra kell fordulni, akkor a baloldali szervomotor sebességét kell visszafogni, a jobboldaliért pedig szinten tartani.

```

if turningright and not forward and not rightturn and not leftturn then
    speed1:= 0.7 * speedIN;
    speed2:=-speed1;
end_if;

```

```

if turningleft and not forward and not rightturn and not leftturn then
    speed2:= 0.7 * speedIN;
    speed1:=-speed2;
end_if;

```

Ez a két programrész egy gyorsított forgást hajt végre, az egyik motort nem csak lelassítja, hanem meg is fordítja a forgásirányát.

```

if not turningleft and not forward and not turningright (*and not turning*) then
    speed1:=0.0;
    speed2:=0.0;
end_if;

```

Ha nincs értelmes parancs, akkor megállítja a motort.

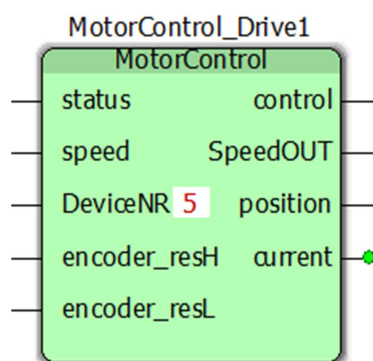
```

if hatra and not forward and not turningright and not turningleft (*and not turning*) then
    speed1:=-0.9* speedIN;
    speed2:=-0.9 * speedIN;
end_if;

```

Ha tolatni kell, akkor a sebesség alapjel 90 %-val forognak hátrafelé a motorok.

A fenti program az utolsó logikai vezérlő, melynek jeleit most már ki kell küldeni a szervo meghajtó modulnak. Erre a feladatra szolgál a „MotorControl” funkcióblokk, melyet fentebb a szervomotor inicializálásával kapcsolatban már megemlítettem.



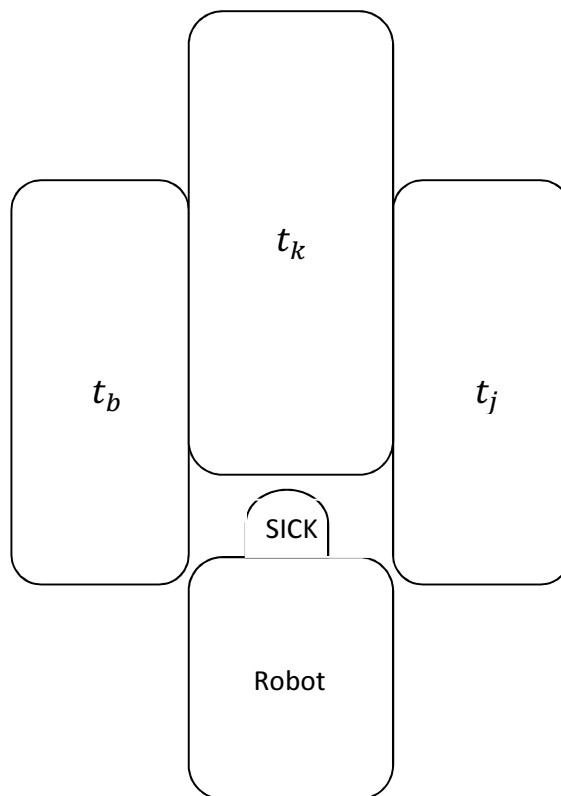
9. 4. ábra

Motor vezérlése

Vezérlőmátrix definiálása

Az előzőekben ismertetett gyakorlati algoritmus mellett van lehetőség elméleti irányítási algoritmus leírására is. Mivel a robot irányválasztása során adott időpillanatbeli tényezők befolyásolják a következő időpillanatbeli irányt és előre nem ismert, hogy a robot előtt hol fog felmerülni akadály, ha egyáltalán felmerül, így az irányválasztási folyamat sztochasztikusnak tekinthető, a robot következő iránya leginkább valószínűségi tényezőkkel becsülhető meg. Az ilyen esetekben alkalmazható a Markov mátrix a következő állapotok megbecsüléséhez.

Ehhez először is meg kell határoznunk az egyes ismert állapotokat. Ezek az állapotok lehetnek: a robot előre, balra, jobbra halad, illetve amikor megfordul kikerülhetetlen akadályok esetén. A Markov mátrix felírásához definiálnunk kell azokat a tényezőket, amelyek befolyásolják az irányválasztást, mint valószínűségi változókat.



10. 1. ábra

Figyelt felületek

Az általunk ismert tényezők egyike az egyes „Field-ek” területe. Általánosságban igaz, hogy a vizsgálat során annál nagyobb a valószínűsége, hogy akadályt érzékel a lézerszkennerek az egyik mezőben, minél nagyobb annak mérete.

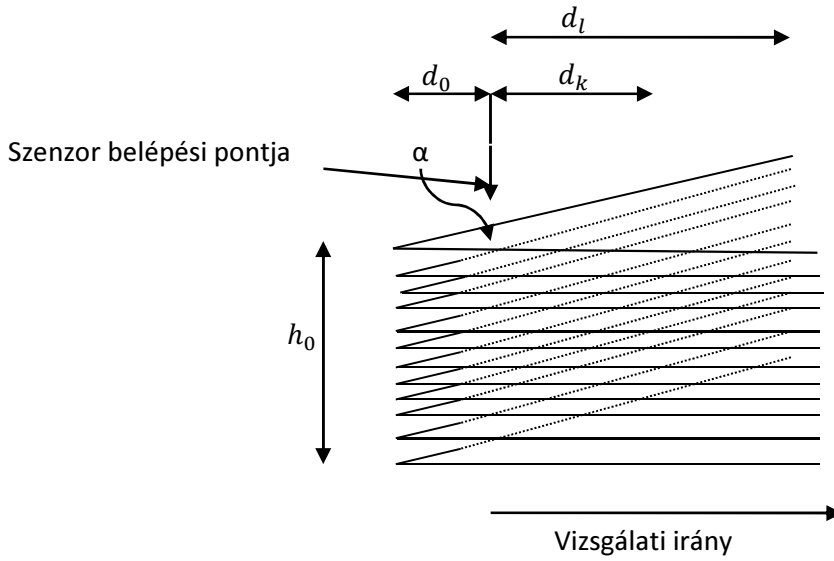
Így rögtön felírható három valószínűséggel arányos mennyiség:

$$P_b \cong \frac{t_b}{t_b + t_k + t_j}$$

$$P_k \cong \frac{t_k}{t_b + t_k + t_j}$$

$$P_j \cong \frac{t_j}{t_b + t_k + t_j}$$

P_b, P_k, P_j változók jelölik rendre a baloldali, a középső, és a jobboldali mezők beütési valószínűségét, abban az esetben, hogyha legalább egy akadály felmerül valamelyik mezőben. A lézerszkennerek kívül még a 3D-s PMD kamera és az ultrahangos távolságérzékelő képes változtatni a robot irányán. Ha ez utóbbi kettő akadályt jelez, valamint ha a lézerszkennerek teljes előtere akadállyal terhelt, akkor a robot képes kimanőverezni ezekből az állapotokból. A 3D-s kamera és az ultrahangos szenzor térben vizsgál, ellentétben a lézerszkennerekkel, ami síkban. Feltételezve, hogy mindkét eszköz α kúpszögű síkmetszetből képezi le a látóterének tartományát, és ez a kúpszelet a szenzor elülső pontjához képest d_0 távolságban el van tolva, az alábbi valószínűségekkel arányos mennyiségeket határozhatjuk meg a kapcsolási távolság (d_k) függvényében:



10. 2. ábra

**Szenzorok mérési
valószínűsége**

Legyen V a szenzorok teljes (működési tartományból adódó) vizsgálati térfogata, legyen A az ehhez tartozó vizsgálati felület!

Ha legalább 1 objektum megjelenik a vizsgálati térfogatban, akkor annak detektálási valószínűsége:

A maximális vizsgálható térfogat:

$$V = \left(\frac{(d_0 + d_l)^2}{4} * \pi * \frac{\alpha}{2\pi} - \frac{d_0^2}{4} * \pi * \frac{\alpha}{2\pi} \right) * h_0$$

$$V = \frac{\alpha}{8} * h_0 ((d_0 + d_l)^2 - d_0^2)$$

Ahol d_l a mérési tartomány felső küszöbe, α pedig a kúpszög radiánban kifejezve.

Így a beállított kapcsolási küszöb szerint a vizsgált térfogat nagysága:

$$V_{vizsgált} = \left(\frac{(d_0 + d_k)^2}{4} * \pi * \frac{\alpha}{2\pi} - \frac{d_0^2}{4} * \pi * \frac{\alpha}{2\pi} \right) * h_0$$

$$V_{vizsgált} = \frac{\alpha}{8} * h_0 ((d_0 + d_k)^2 - d_0^2)$$

behelyettesítve:

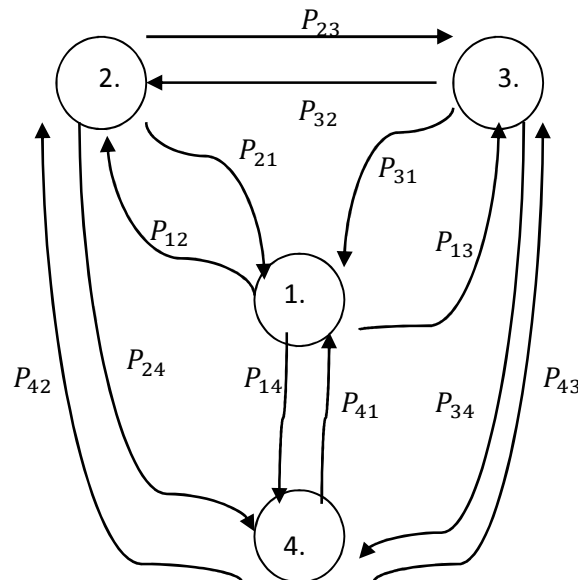
$$P_{kamera}, P_{uh} \sim \frac{V_{vizsgált}}{V} = \frac{((d_0 + d_k)^2 - d_0^2)}{((d_0 + d_l)^2 - d_0^2)}$$

egyszerűsítve:

$$P_{km}, P_{uh} \sim \frac{V_{vizsgált}}{V} = \left(\frac{d_k}{d_0}\right)^2 * \frac{1 + 2 * \frac{d_0}{d_k}}{1 + 2 * \frac{d_0}{d_l}}$$

Ebből látható, hogyan függ a kapcsolási küszöbszinttől a detektálási valószínűség. Mivel ez egy domináns valószínűség lesz, mint később látni fogjuk, így nem célszerű túl nagyra választani az értékét.

Az irányválasztás szempontjából 4 állapot létezik, így az egyes átmeneteket leíró gráf így adható meg:



Az fenti gráf szemlélteti az egyes állapotok közötti átmenetek valószínűségét. Most már fel lehet írni a Markov mátrixát a vezérlésnek:

$$\begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \\ P_{41} & P_{42} & P_{43} & P_{44} \end{bmatrix}$$

Ahol a vízszintes sor jelöli rendre az 1,2,3,4-es állapotokat, amik a robot adott pillanatbeli irányát írják le. A függőleges oszlopok jelölik a következő állapotot. Tehát P_{23} megadja, hogy mekkora a valószínűsége, hogy a robot balfordulatból jobbfordulatba vág át.

Állapotok:

- 1.: Robot előre halad
- 2.: Robot balra fordul
- 3.: Robot jobbra fordul
- 4.: Robot megfordul

Gyakorlati szinten az alábbi gondolatmenet szerint lehet meghatározni az egyes események bekövetkeztének valószínűségét:

Ha a robot 1,2,3 állapotokban van, akkor ebből bármikor kimozdítható, hogyha a kamera, vagy az ultrahangos érzékelő jelez, vagy mindhárom mezőben akadály lép fel.

A robot az egyenes irány mozgását csak akkor szakítja meg, ha a fenti esemény bekövetkezik, vagy hogyha az elülső mezőben akadály lép fel. Tehát ha vagy a bal, vagy a jobb mezőben lesz akadály attól még a robot nem változtatja meg az irányát.

Így $P_{11} = P_b + P_j$ és mivel ez az állapot az alapértelmezett, így bármely másikból ebbe az alapállapotba tér vissza a robot, tehát nem fordul tovább egyik irányba sem.

Ha a robot előtti térbe akadály kerül, akkor attól függően, hogy tőle jobbra, illetve balra van akadály eldönti, hogy melyik irányba fog elfordulni. Így akkor fordul jobbra, hogyha előtte és balra is akadály van, illetve akkor fordul balra, hogy tőle jobbra és előtte is van akadály.

Az hogy jobbra való forgásból hirtelen balra forduljon , vagy fordítva az nem fordulhat elő, ugyanis a forgás után mindig egy időpillanatra előre halad a robot.

Behelyettesítve a mátrixba:

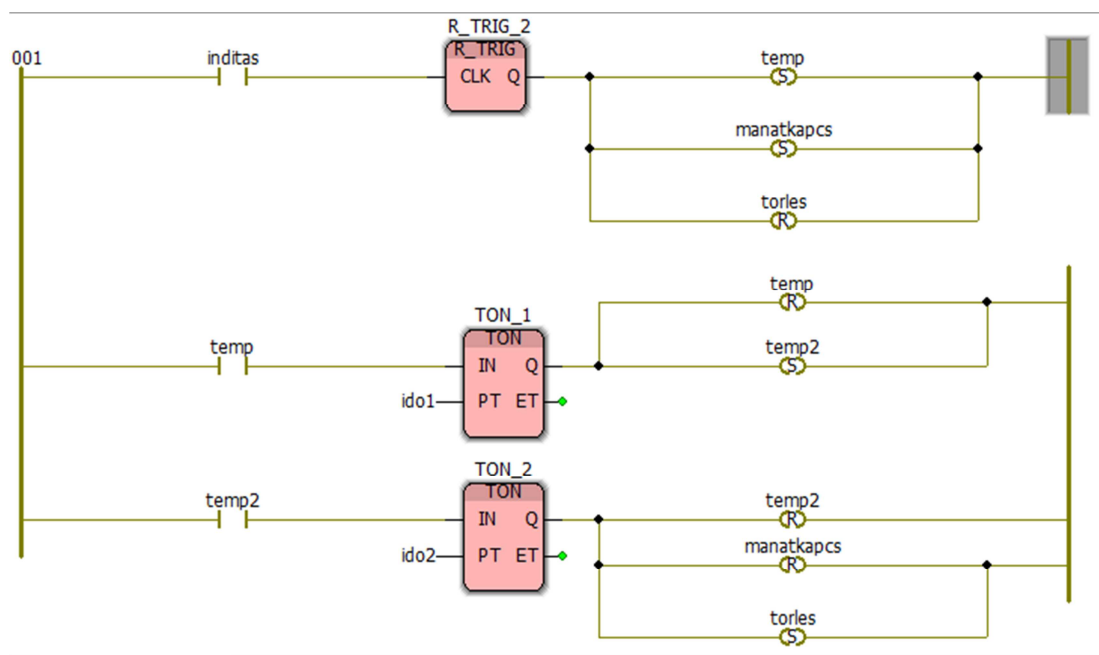
$$\begin{bmatrix} P_b + P_j & P_k * P_j & P_k * P_b & P_k * P_b * P_j + P_{uh} + P_{km} \\ P_b + P_j & P_k * P_j & 0 & P_k * P_b * P_j + P_{uh} + P_{km} \\ P_b + P_j & 0 & P_k * P_b & P_k * P_b * P_j + P_{uh} + P_{km} \\ P_b + P_j & 0 & 0 & 0 \end{bmatrix}$$

Ebből látható, hogy sem az ultrahangos szenzor, sem a kamera kapcsolási küszöbszintjét sem szabad kicsire állítani.

Speciális esetek:

Mint említettük, az auto/manuális átkapcsolásnak az egyik előnye, hogy a lézerszenzor alapján történő vezérlést felül tudjuk bírálni egy saját programrész szerint. Ezen programrész neve „manover”, melynek feladata, hogy a robotot hátra vezérelje, majd 180 fokkal megfordítsa.

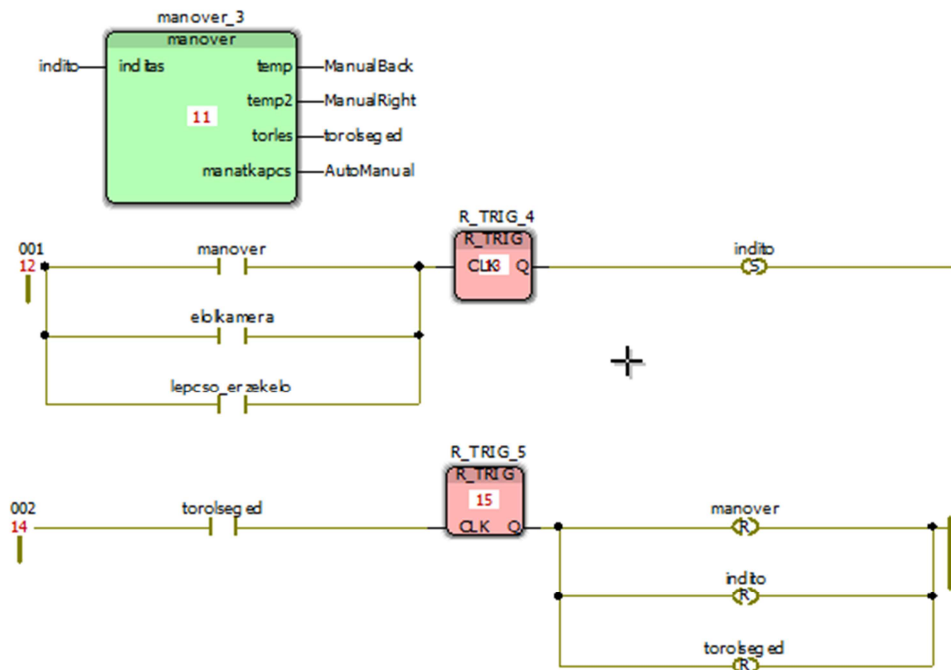
A programot létra nyelven írtuk meg melynek forrása az alábbi:



11. 1. ábra

Manőverező program

A program 2 szekvenciát hajt végre, melynek lefutása késleltetve bekapcsoló (TON) időzítőkkal állítható.



11. 2. ábra

Manőver meghívása

Mint látható a manőverezést 3 esemény VAGY kapcsolatának felfutó éle indítja el. Ez a három esemény az alábbi lehet:

- A lézerszkennelőről mindhárom FIELD (jobb oldali, bal oldali, elülső) akadályt tartalmaz. Ennek tipikus esete, amikor a robot egy sarokba kerül, és nincs más megoldás, mint hogy kivezéljük onnan.
- A 3D-s kamera maga előtt akadályt észlelt, melynek kikerülése már nem lehetséges, hiszen a kamera magasságában a lézerszkennelőről nem lát, továbbá a kamera látószöge kellően kicsi ahhoz, hogy jól elkülöníthető lehessen az előtte kis távolságban levő akadály.
- Továbbá, ha a saját fejlesztésű lépcsőérzékelő modul jelzi, hogy lépcső van a robot előtt. Ebben az esetben sincs min gondolkodni, tolatni kell, és megfordulni.

Nyomkövetés lézerszkennel segítségével

Ennek a feladatnak az a lényege, hogy elérjem, hogy a mobil robot saját intelligenciáját felhasználva, az esetleges akadályokat figyelembe véve kövessen egy a környezetétől eltérő tulajdonságú tárgyat.

Ez a tulajdonság a fényvisszaverő képesség lehet jelen esetben, hiszen a lézerszkennel nem csak az akadályfigyelésre, hanem a visszaverődő sugarak intenzitásáról is tud információt szolgáltatni. Ez az adat az RSSI adat.

Ahhoz, hogy a kiszemelt céltárgyat a robot követni tudja, sokkal nagyobb, vagy sokkal kisebb fényvisszaverő képességű anyagból kell készülnie. Mivel egy hagyományos környezetben több olyan anyag/felület van, amelyik nem túl jól veri vissza hullámokat, így jobb megoldás egy nagyon jó fényvisszaverő „macskaszem” alkalmazása. A „macskaszem” egy kartonhenger, ami nagyon jó fényvisszaverő anyaggal van bevonva.



12. 1. ábra

„Macskaszem”

Az eddigi feladatok során a lézerszkennel felparaméterezéséhez a gyári SOPAS nevezetű szoftvert használtuk, és szimplán BOOL bemeneti biteken keresztül kaptunk információt a Field-ek foglaltságáról. Mivel sokkal részletesebb adatra van szükségünk, így Etherneten keresztül TCP/IP protokollt használva tudjuk a lézerszkennel mérési eredményeit lekérdezni.

Mivel a mobil robot tartalmaz egy ipari PC-t és a 3D-s kamera képének feldolgozásához is Matlab programot használtunk, így logikus, hogy integráljuk össze a kamera vezérlőt és a lézerszkennel vezérlőt. Ez megoldható, így értelemszerűen a lézerszkennel vezérlését is Matlabból ajánlott megoldani. Meg lehetne oldani PLC-s környezetből is a kommunikációt, de a Matlab-os környezet leegyszerűsíti a probléma megoldását.

A Matlab képes TCP/IP kommunikációt megvalósítani, melyet magas szintű fájlkezeléshez hasonlóan lehet kezelni a programból.

A felhasznált program ismertetése:

ipconnect:

```
function [hnd] = ip(ip,port,buffertype,buffersize)
    hnd=tcip(ip,port,buffertype,buffersize);
    fopen(hnd);
end
```

A programrész meghívja a Matlab gyári „tcpip” függvényét, ami létrehoz egy TCP/IP objektumot. Ennek az objektumnak meg kell adni paraméterként az IP címet, port számot, a buffer méretét és típusát. Ezek paraméterek később kerülnek kifejtésre. A függvény visszaad egy handlert, amit a későbbi műveletekhez fogunk használni.

ipsend:

```
function [status] = ipsend(hnd,payload)
    fwrite(hnd,2);
    fwrite(hnd,payload);
    fwrite(hnd,3);
    status=0;
end
```

Ez a függvény egy olyan teljes telegramot állít össze, amit a lézerszkenner igényel. Az STX(2) karakter után sztringként kell megadni a parancsot, majd lezárásául ETX(3) karaktert kell kiküldeni. Ezen karakterek a sztenderd ASCII táblából származnak.

ipreceive:

```
function [data] = ipreceive(hnd,lenght)
    data=fread(hnd,lenght);
end
```

Ez a program megadott hosszúságú adatot olvas ki a TCP/IP vételi oldali bufferjából.

sendtelegram:

```
function [receivedata,db]=sendtelegram(hnd,payload)
    ipsend(hnd,payload);
    while hnd.BytesAvailable == 0
    end;
    db=hnd.BytesAvailable;
    receivedata=char(ipreceive(hnd,hnd.BytesAvailable));
end
```

Ez a program használja az eddig ismertetett részprogramokat. A megadott sztring parancsot STX, ETX karakterekkel kiegészítve kiküldi a lézerszkennerek felé. Ezt követően vár, amíg válasz nem érkezik, majd a visszajelzett darabszámú bájt kiolvassa, majd a bináris adatot ASCII karakterekké alakítja át és visszaadja a hívó függvénynek.

Ezen függvényeket csak akkor tudjuk használni megfelelően, ha ismerjük a lézerszkennerek protokollját.

Lézerszkennerek paraméterezésének menete:

- Kapcsolódás a lézerszkennerekhez
- Bejelentkezés a lézerszkennerekhez megfelelő jogosultság kiválasztásával
- Mérési adatok konfigurálása
- RUN módba való belépés
- Mérés triggerelése
- Mérések befejeztével a lézerszkennerek STANDBY módba való léptetése

Kapcsolódás a lézerszkennerekhez:

```
hnd=ipconnect('192.168.0.11',2112,'INPUT',4096);
```

Meg kell adni, hogy mi a lézerszkennerek IP címe, illetve port címe, a buffer típusa bemeneti, hiszen adatot olvasunk be, a méretét 4 kbyte-ra állítottam, ez biztosan elég lesz.

Sikeres kapcsolatfelvételt követően a Workspace menüben megjelenő „hnd” változót megnyitva a státusz mezőben „open” állapotot kell visszajeleznie. Ezután a „hnd” változó fog utalni a felépített kapcsolatra.

Telegram	Description	Variable	Length	Values ASCII	Values Binary
Command Type	Sopas by name	String	3	sMN	73 4D 4E
Command	User level	String	13	SetAccessMode	53 65 74 41 63 63 65 73 73 4D 6F 64 65
User level	select user level	Int_8	1	02 maintenance 03 authorised client 04 Service	
Password	"Hash" - value for the User level	Uint_32	4	00 00 00 00h – FF FF FF FFh	

Userlevel	Password	Hash-Value
Maintenance	main	B21ACE26
Authorised client	client	F4724744
Service	servicelevel	81BE23AA

12. 2. ábra

A fenti ábrán a lézerszkennerek paraméterezési könyvének egyik szemelvénye, mely megadja, hogy milyen paraméterekkel tudunk a megfelelő jogosultsági szintre bejelentkezni.

A „szerviz” jogosultsági szint a legmagasabb, ebbe belépve biztosan tudunk állítani minden paraméterét a lézerszkennereknek:

```
sendtelegram(hnd,'sMN SetAccessMode 04 81BE23AA');
```

Ezután a lézerszkennerek megadjuk, hogy milyen adatokat küldjön vissza egy mérés után:

```
sendtelegram(hnd,'sWN LMDscandatacfg 01 00 1 0 0 00 00 0 0 0 +1');
```

Az LMDscandatacfg szolgálat a mérési eredmény konfigurálására az alábbiak szerint:

Adat csatorna	2 bájt „01 00” -> Output 1-es csatorna
RSSI bekapcsolása	1 bájt „1”
RSSI felbontás	1 bájt „0” -> 8 bites felbontás
Adategység digitben	1 bájt „0”
Enkóder adat	2 bájt „00 00” -> nincs enkóder használatban
Pozíció érték csatolása	1 bájt „0” -> nincs szükségünk rá
Eszköz név csatolása	1 bájt „0”
Komment csatolása	1 bájt „0”
Időpont adat csatolása	1 bájt „0”
Kiolvasási gyakoriság	2 bájt „+1” -> minden mérési eredményt

A beállítás érdemi lényege a 8 bites RSSI csatorna bekapcsolása, illetve az Output1-es kimeneti csatorna bekapcsolása.

Alapesetbe a lézerszkennerek 0.5 °-ként készít mérési eredményt, és mivel 270 fokos a látószög, így 540 mérési adatot várunk vissza.

1 mérés indításához az alábbi parancsot kell kiküldeni:

```
data=sendtelegram(hnd,'sRN LMDscandata');
```

A visszakapott 4096 db bájt a paraméterezési könyv „Send data permanent” címszáva alatt tudjuk kiolvasni. Ez a vektor nagyon sok információt tartalmaz, de ebből számunkra a jelen feladatban 540 bájtra van szükségünk.

Outputchannel 1..4 (16 bit)	Content	Defines the Content of the Output channel	String	5	LMS1xx: DIST1: radial Values of first pulse RSSI1:Energy Values of first pulse DIST2: radial Values of 2nd pulse RSSI2:Energy Values of 2nd pulse LMS5xx: DIST1 DIST2 DIST3 DIST4 DIST5 No RSSI Values	44 49 53 54 31 52 53 53 49 31 44 49 53 54 32 52 53 53 49 32
	Scale factor		Real	4	00000000h - FFFFFFFFh	
	Scale factor offset	LMS = 0	Real	4	00000000h - FFFFFFFFh	
	Start angle	Output format : 1/10.000°	Uint_32	4	LMS1xx: -450.000 +2250.000 LMS5xx: -50.000 +1850.000	
	Steps	Output format : 1/10.000°	Uint_16	2	LMS1xx: 1000 10.000 LMS5xx: 1667..10.000	
	Amount of Data	Defines the number of items om measured output	Uint_16	2	0000h – FFFFh	
	Data_1 Data_n	Data stream starting Data_1 to Data_n	Uint_16	2	0000h - 4E20 (LMS100) C350 (LMS150) FDE8 (LMS 1xx without limit)	

12. 3. ábra

Adatformátum

A fenti képmetszet mutatja, hogy van egy olyan mező, ami jelzi az RSSI1 adat kezdetét, ez headerként funkcionál. Az „RS” karaktersorozat után a mérési eredmény az 'R' indexe után 42 karakterrel később érkezett. Ennek oka, hogy az adatot elválasztó <space> karaktereket is le kell számolni.

Ennek megfelelően a mérési eredményeket is <space> karakterrel elválasztva fogjuk megtalálni. Az eredményeket ASCII karakterek formájában kapjuk meg, de hexadecimális kódolásban. Ahhoz, hogy ezekből a karakterekből megkapjuk a decimális Remission értéket, az alábbi algoritmust használtam:

```

if(tomb(j+42+k)>= 'A' && tomb(j+42+k) <='F') rssi=(tomb(j+42+k)-65+10)*16; else
rssi=(tomb(j+42+k)-48)*16; end

k=k+1;
if(tomb(j+42+k) ~= '')

    if(tomb(j+42+k)>= 'A' && tomb(j+42+k) <='F') rssi=rssi+(tomb(j+42+k)-65+10)*1;
else rssi=rssi+(tomb(j+42+k)-48)*1; end

    k=k+1;
end

if(tomb(j+42+k) == '') z=z+1; meres(z)=rssi; k=k+1; end

```

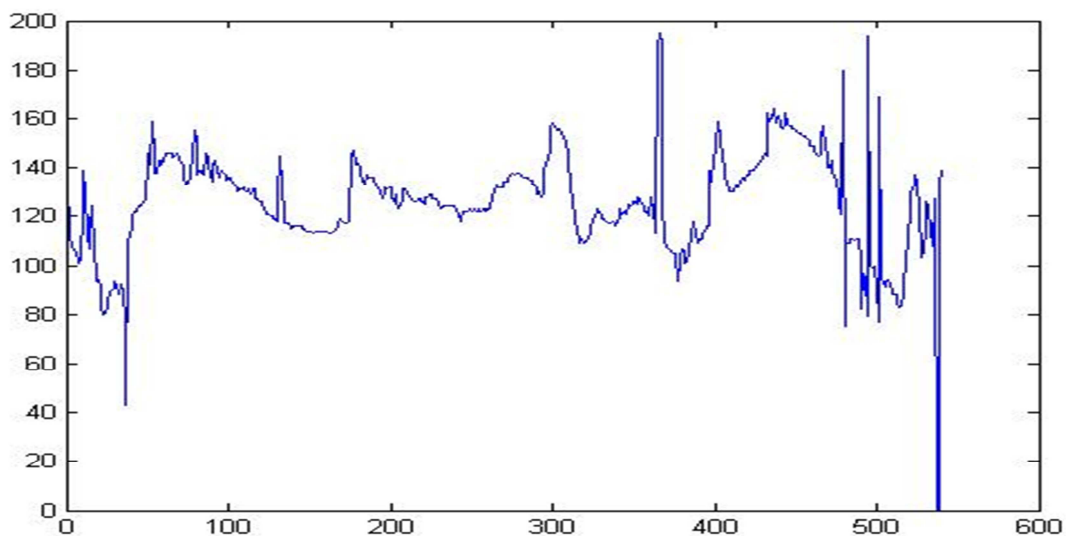
Ez a kódrészlet azt mutatja meg, hogy hogyan kell ASCII karakterekből az eredeti decimális értéket visszaállítani. Bájt folytonosan haladva a nagyobb helyi értékű fél bájtot kapjuk.

Abban az esetben, hogyha a karakter nagybetűs, akkor a 65-ös offset értéket kell kivonni, ha kisbetűs, akkor a 48-as offset címet kell kivonni az értékből.

A nagyobb fél bájt értéke 16 X-os, az alsó fél bájt 1 X-es súllyal érvényesül.

Az adatbájtok előtt 2 bájtton, de 3 fél bájtton kapjuk meg azt, hogy hány darab mérési eredményt kell feldolgoznunk.

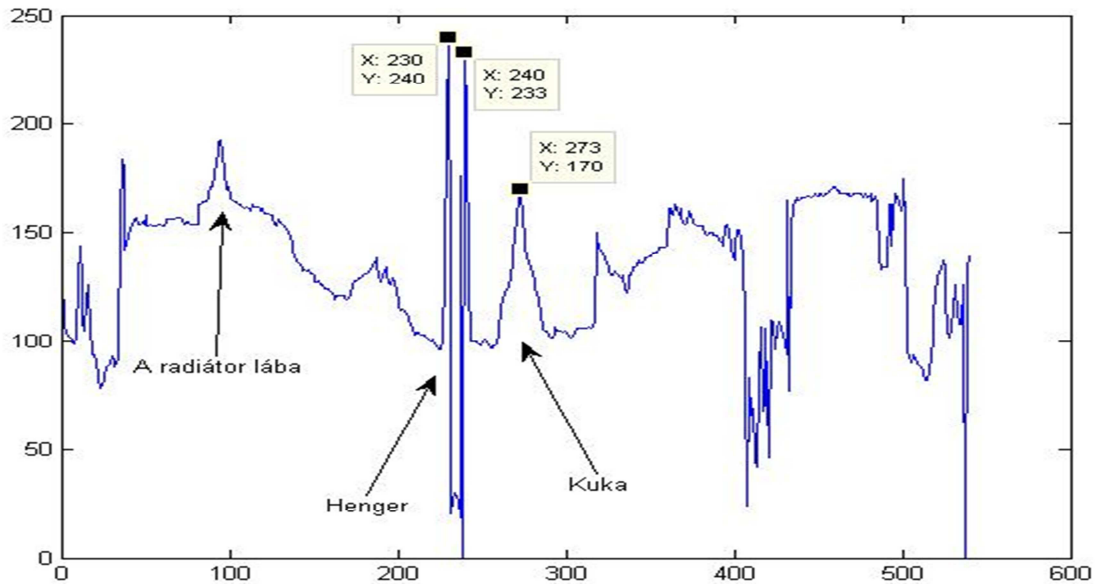
Ha a mérési eredményeket egy tömbbe letároljuk, és megjelenítjük, akkor megfigyelhetjük, hogy mit lát a lézerszenzor a környezetéből:



12. 4. ábra

Zavarral terhelt RSSI kép

Az RSSI értékeket 8 biten kapjuk vissza, tehát 255-a maximum érték. Mivel a fényvisszaverő prizma nem volt lehelyezve, így a 200-as érték fölé nem kaptunk mérési pontot.

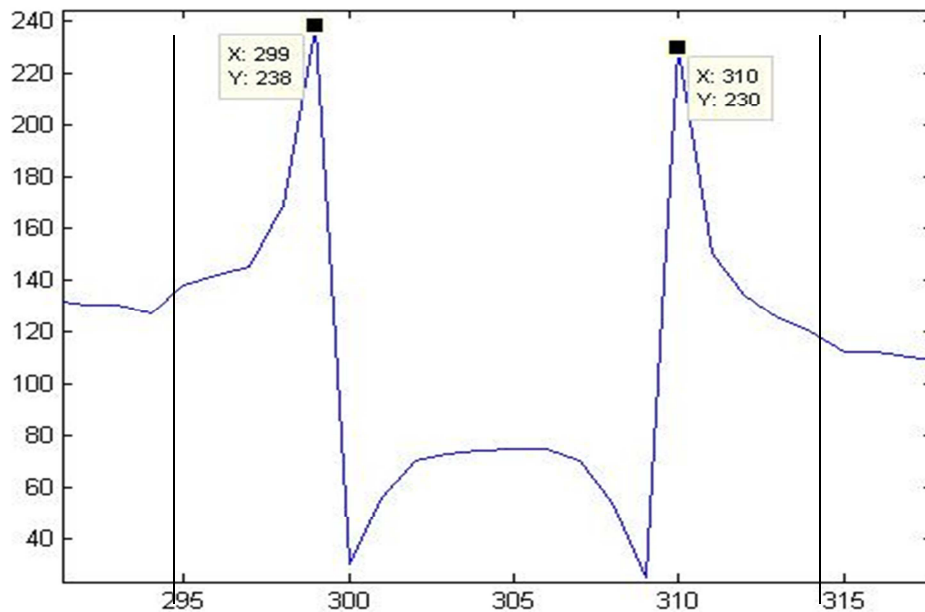


12. 5. ábra

Hasznos mérési eredmény

A következő mérési eredmény már a kartonhengerből kialakított prizmát is tartalmazza, illetve szándékos zavarásként egy kukát is a „macskaszem” mellé tettünk. A kuka alsó felén egy fémcsík található, ami jó fényvisszaverő.

A hengeres kuka esetében jól látható, hogy a fényt érintőirányra merőleges irányban veri vissza, tehát a középpontjától a szélek felé haladva nem a fényforrás irányába veri vissza a sugarakat, hanem szétszórja azokat. Az elvárásoknak megfelelően a legnagyobb Remission értékű mérési pontok a „macskaszemet” azonosítják, de kinagyítva az ábrát érdekes eredményt kaptam:



12. 6. ábra

A macskaszem kinagyított RSSI képe

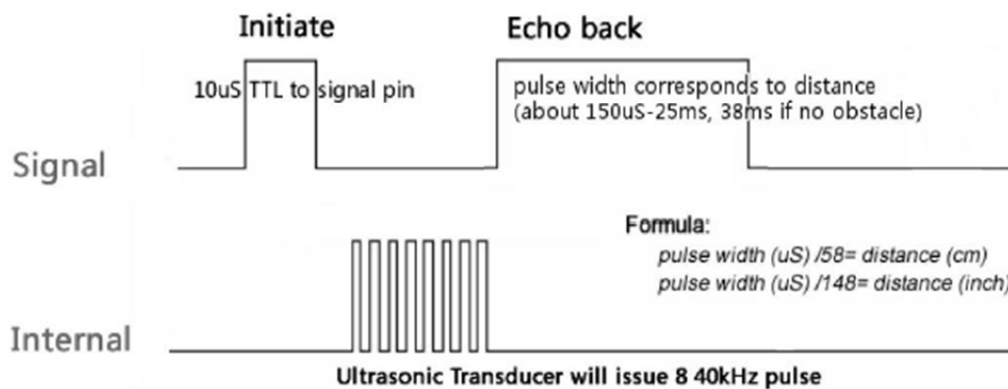
Az ábrán látható, hogy a henger lézerszkennerhez legközelebbi pontjáról verődik vissza a legkevesebb fény. Ennek oka a fényvisszaverő fólia szerkezetéből fakadhat. A fóliában a fényvisszaverő szemcsék nem a fólia felületével párhuzamosan helyezkednek el, hanem rézsút 45 fokban, így amikor a köralak érintője kb. 45 fokos, akkor kapjuk a maximális visszaverődést.

A mérési eredmények alapján megállapítottuk, hogy a fényvisszaverő „macskaszem” megtalálható a maximumérték figyelésével. A program, amit írtam meghatározza, hogy mely mérési eredményhez mekkora maximum tartozik. Mivel a lézerszkennek 0-tól 270 fokig mér, így pontosan a 270-es indexű elem mutat rá a robot előtti pontra, a 270-es index alatti elemek a robottól balra, a nagyobbak tőle jobbra helyezkednek el. A megfelelő irányítottság érdekében 15 fokos holtzónát állapítottam meg, ettől a tartománytól balra és jobbra levő tárgyak az egyenes útmenetből el fogják irányítani a robotot, de csak abban az esetben, hogyha a lézerszkenner nem detektál akadályt maga előtt.

A feladat kiértékeléseképp megállapítható, hogy a kitűzött célt elértük, a robot a tőle távolban elhelyezett prizmat megtalálja és le is követi.

Ultrahangos lépcsőérzékelő

A robottal való foglalkozásom alkalmával szembetűnő volt a robot védtelensége esetlegesen felmerülő mélyedésekkel, lefelé irányuló lépcsővel szemben. Saját ötletemként felmerült, hogy a meglévő mikrokontroller programozói, NYÁK tervező - és gyártó tapasztalatomat felhasználva készítsék egy egyszerű eszközt, amellyel csökkenthető a robot leeséséből adódó kockázat. Mivel az eszköz teljesen saját fejlesztésű, így a boltban kapható alkatrészek határozták meg elsődlegesen a tervezési szempontokat. A lépcsőérzékelő lelke egy ultrahangos szenzor, mely a trigger bemenetére érkező impulzus hatására 8 db 40 KHz-es négyszögimpulzust ad rá az ultrahangos sugárzóra. A sugárzóból kilépő hullámok a tárgy felületéről reflektálódnak, és egy ultrahangos vevőegysége jutnak. Az általam használt ultrahangos érzékelő beépített intelligenciával rendelkezik, ugyanis nem a vevőegységbe visszajutó impulzusokat küldi tovább a mikrokontrollernek, hanem egy impulzust, aminek a hossza arányos a visszaverődési késleltetéssel idővel, magyarul a tárgytávolsággal.



13. 1. ábra

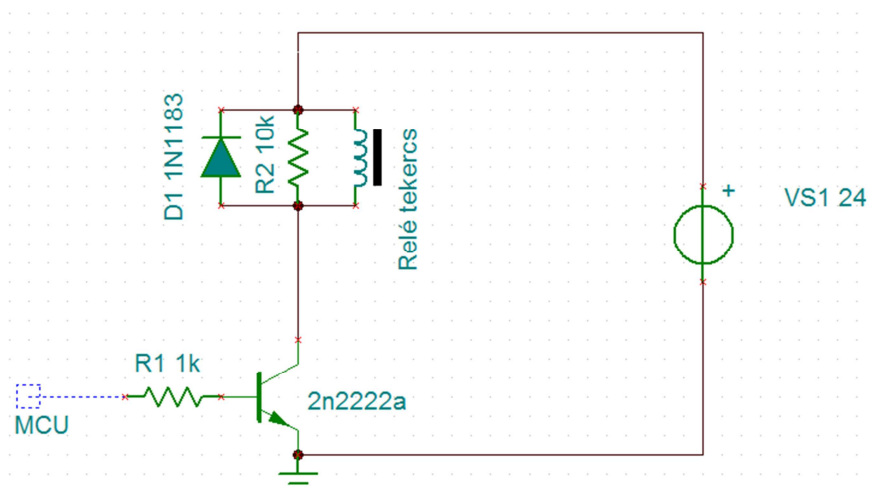
Szenzor jelformái

$$\text{Távolság (cm)} = \frac{\text{Impulzusszélesség(us)}}{58}$$

A fenti formulából is látható, hogy a távolság arányos a visszakapott impulzus szélességével.

Az ultrahangos szenzor kezeléséhez egy 8 bites PIC mikrokontrollert használtam, amely a TTL feszültség szintű portjával vezérli a szenzort. Mint a fenti képen is látható, egy 10 us-os impulzust adunk a szenzor Signal portjára, mely trigger jelként funkcionál. Ezt követően átállítva a portunk irányát bemenetire, lemérjük a visszakapott impulzus hosszúságát. Az impulzus hosszúságából felesleges visszaszámolni a távolságot lebegőpontos formátumúba, hiszen ez a 8 bites MCU-t feleslegesen leterhelné, így a TIMER regiszterben lévő értéket egy arányossági tartományban megfeleltetve komparáljuk az általunk előzetesen kiszámított értékkel. Tapasztalati úton 15 cm-es távolságot határoztunk meg, melyhez kb. 900 us hosszúságú impulzusnál hosszabb hosszúságú impulzust kell visszakapjunk, hogy a készülékünk a PLC-nek visszajelezzon figyelmeztetésképpen. A távolság meghatározásánál figyelembe kellett venni, hogy az útfelület egyenetlen, így a kapcsolási távolság meghatározásánál ráhagyással kellett meghatározni a küszöbértéket.

A visszajelző áramkör úgy lett kialakítva, hogy egy relé bemeneti tekercsét gerjessze, így a relé kimeneti kontaktusa fogja meghajtani a PLC optocsatolt bemenetét:



13. 2. ábra

Behúzó tekercs kapcsolása

Az elrendezés előnye lett volna, hogy a robot jóval a kerekei előtt érzékelni tudta volna a problémát, így biztosan lett volna ideje a manőverezésre, viszont az ábrán is látható, hogy a nem merőleges felületről alig jut vissza jel a szenzorba, így egy távoli reflektáló tárgyról visszaérkező hullám lesz a legmeghatározóbb, ami a mérési eredmények alapján hasztalanná tenné az szenzort, alkalmatlan lenne a süllyedések felismerésére.

A megoldást az jelentette, hogy az úttal merőleges kapcsolatot hoztam létre a szenzor és az útfelület között. A robot legelőrébb elhelyezkedő pontja a lézerszkenner, így adódott a gondolat, hogy a lézerszkennert tartó kar felfüggesztési pontjára egy fémlemezt szereljek fel, és ennek a fémlemeznek a legvégére helyezzem el a lépcsőérzékelőt.

Tartalomjegyzék

1. Bevezető.....	1. oldal
2. Szervo vezérlés.....	9. oldal
3. OPC szerver.....	16. oldal
4. IFM O3D200-as kamera.....	20. oldal
5. Gyári konfiguráló szoftver.....	24. oldal
6. O3D201 kamera képfeldolgozása	26. oldal
7. SICK LMS100 lézerszkennel bemutatója	28. oldal
8. SICK lézerszkennel kezelése.....	32. oldal
9. A PLC-n futó vezérlőprogram.....	33. oldal
10. Vezérlőmátrix definiálása	39. oldal
11. Speciális esetek.....	45. oldal
12. Nyomkövetés lézerszkennel segítségével.....	47. oldal
13. Ultrahangos lépcsőérzékelő	55. oldal