Debreceni Egyetem
Informatikai Kar

# Robotvezérlés telekommunikációs eszközökkel

Témavezető:
dr. Szabó István
egyetemi docens

Készítette:
Juhász Ádám
Mérnök-informatikus

Debrecen
2008

# Tartalomjegyzék

# Bevezetés

A robotika manapság az életünk szinte minden területén megjelenik, legyen az iskolai oktatás, gyártási folyamatok, szórakoztató ipar, vagy a mindennapi életet megkönnyítő háztartási gépek. Minden bizonnyal egyre nagyobb szerepet játszanak majd a jövőben, így kézenfekvő és szimpatikus témának bizonyult a szakdolgozatom témájának megválasztásakor.

A rendelkezésemre álló eszköz sokféleképpen megépíthető és programozható. Ha ezeket a tulajdonságokat kibővítjük a telekommunikáció által nyújtott lehetőségekkel, akkor egy sokoldalú, sok területen alkalmazható robotot kapunk.

Pár mondatban megfogalmaznám a robot fogalmát.
A robot teljesen független gép, amely képes megváltoztatni a környezetet, amelyben működik.
Újraprogramozható, többfunkciós manipulátoranyagok, eszközök, részegységek mozgatására, megváltoztatására programozott mozdulatsor segítségével különféle feladatok elvégzése érdekében. (*Robot Institute of America, 1980)*

A robotika pedig az érzékelés és az akció közötti intelligens kapcsolat.

Mivel szorosan a témához kapcsolódik, szeretném megemlíteni a robotika Isaac Asimov által megfogalmazott három alaptörvényét:

1. *A robotnak nem szabad kárt okoznia emberi lényben vagy tétlenül tűrnie, hogy emberi lény bármilyen kárt szenvedjen.*
2. *A robot engedelmeskedni tartozik az emberi lények utasításainak, kivéve, ha ezek az utasítások az első törvény előírásaiba ütköznének.*
3. *A robot tartozik saját védelméről gondoskodni, amennyiben ez nem ütközik az első és a második törvény előírásaiba.*

A robot programozása számítógépről történik. Írhatunk olyan programot is, amely a gépről futtatva küld vezérlőjeleket a robotnak, és írhatunk olyat is, amelyet letöltünk a robotra, és a számítógéptől függetlenül futtatjuk.

Ezen dolgozat célja, hogy a fent említett sokoldalúsággal egy Lego Mindstorms robotot felruházzunk, feltárjuk a programozási lehetőségeket és részletezzük ennek megvalósításait, valamilyen szempontok alapján összehasonlítsuk az alkalmazható programozási nyelveket. Konkrétabban egy olyan önállóan működő rendszer megalkotása, amely egy robotot egy kezdő koordinátáról el tud navigálni egy megadott cél koordinátára. A robot koordinátáinak megállapítására kamerát használok, amelyet egy szerverprogram működtet, és a képkockákból megállapított adatokat egy navigáló programhoz juttat el. A navigáló program feladata pedig az, hogy Bluetooth csatornán keresztül vezérlőjeleket küldjön a robotnak a feldolgozott koordináta adatoknak megfelelően. Továbbá a robotnak képesnek kell lennie akadályokat elkerülni olyan módon, hogy ha akadályt vesz észre maga előtt, akkor felülbírálja a navigátor program által kikalkulált útvonalat és kikerüli az akadályt, ugyanakkor nem hiúsul meg az eredeti cél elérése sem.

A feladat megoldása során az NXT-G, RobotC, Mindsqualls nyelveket és programkönyvtárakat tanulmányoztam részletesebben, ezekre a dolgozat későbbi részeiben részletesen kitérek, több más hasonló nyelv elemzése mellett.

A fejlesztésekhez az NXT-G és RobotC saját fejlesztői környezetét, valamint a Microsoft Visual Studio-t használtam. A feladatban nagy szerepet játszó Bluetooth kommunikáció működési elveit és a különböző programnyelvekben való alkalmazási módjait az internetről szerzett források alapján igyekeztem elsajátítani. Ezek közül a leghasznosabb információk szintén megemlítésre kerülnek a dolgozat későbbi részeiben.

# Egy robot részei

## A mechanikai rendszer

Szükséges, hogy legyen egy *mechanikai rendszer*, ez a robot azon részrendszere, amely az akciót valósítja meg [1]. Az akció során szükség lehet a robot mozgatására a környezetben, ezt a *helyváltoztató berendezés* végzi. Motorok, különböző mechanikai tagok teszik lehetővé a helyváltoztatást. Az alábbi képeken két példát láthatunk helyváltoztató berendezés megvalósítására.


1. ábra. Robotláb és kerekek

A helyváltoztató képesség célja gyakran az, hogy bizonyos objektumokat képes legyen a robot megközelíteni, tudjon velük operálni, vagy rajtuk méréseket elvégezni. Ez a *mechanikai rendszer* feladata, szenzorok, érzékelők és beavatkozók segítségével.

# A szenzoros és irányító rendszer

A *szenzoros rendszer belső állapot*a maga a mechanikai rendszer állapota, míg a *külső állapot* a környezet állapotát jellemzi.
Sokféle külső környezeti állapot létezik. Ahhoz, hogy a különböző környezeti tényezőket (pl. hőmérséklet, fényerősség, mágnesesség stb.) láthatóvá és érzékelhetővé tegyük a robotunk számára, fel kell szerelni a megfelelő szenzorokkal.

Ezek közül néhány:

- Fényszenzorok (infravörös ledek, fototranzisztorok-diódák, foto reflektorok)
- Hall szenzorok (mágnesesség)
- Vibrációs szenzorok (oszcilláció és rezgés érzékelésére)
- Légnyomásváltozás szenzor
- Ultraszonikus szenzorok (ultrahang kibocsátás útján távolságot határoz meg)
- Pyro szenzorok (sugárzást érzékelik)

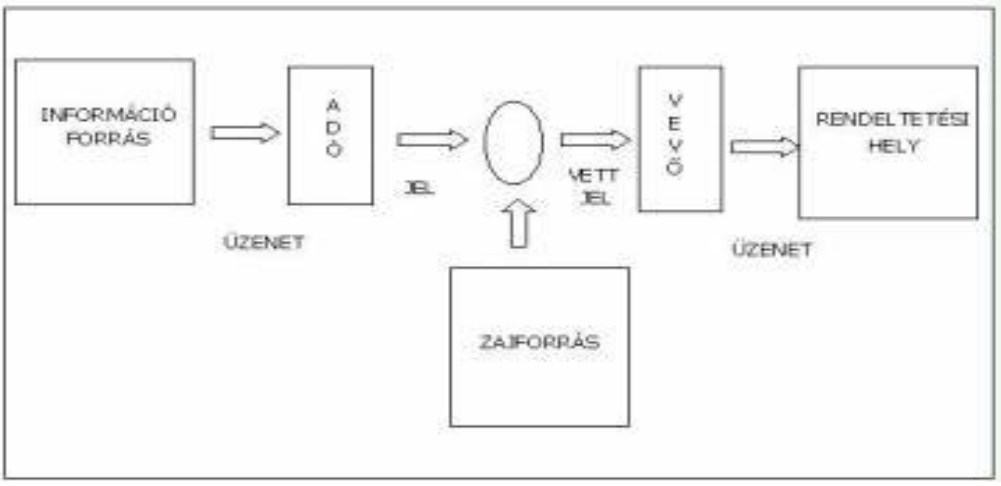
Ezeken kívül még nagyon sokféle szenzor létezik, talán ezeknél alapvetőbbek is (pl. nyomógomb, mikrofon), ezekről a későbbiekben még lesz szó a konkrét feladatomban felhasznált eszközök kapcsán.

Az akciót és érzékelést az *irányítórendszer* hidalja át, a mechanikai rendszer és a környezet által jelentett korlátozásokat figyelembe véve mérlegelni kell, és döntést kell hozni. Ez történhet részben vagy egészben emberi irányítással, vagy a robot mesterséges intelligenciájával önállóan is cselekedhet.

# Telekommunikáció

A *telekommunikáció* tulajdonképpen olyan helyzetekben ad lehetőséget kommunikációra, amikor az adó és a vevő fél nem azonos helyen tartózkodnak, és ezt nem az információ eredeti hordozójának áthelyezésével teszi lehetővé. Jellemzően elektronikai eszközök segítségével kommunikációt, esetleg adatátvitelt lehet megvalósítani (távíró, telefon, stb.).

A megvalósításhoz szükséges az *adó*, egy *átviteli közeg, csatorna*, és a *vevő*. A csatornának megfelelően az adónak az üzenetet egy átvihető jellé kell kódolnia, amelyet az átviteli közeg továbbít, majd a vevő visszaalakítja a vett jelet. Gyakran előfordul, hogy a jel továbbítása veszteségekkel jár, ilyenkor a vevő nem tudja garantálni a tökéletes dekódolást. Ez a veszteség azért van, mert a csatornákon *zaj* van és véges a *sávszélesség*ük. Ezen tények kiküszöbölésére sok tömörítési, kódolási, és hibajavítási algoritmus, tétel jött létre.




2. ábra. Kommunikáció részei

Mivel elektromos eszközök teszik lehetővé a telekommunikációt, szükséges, hogy legyenek olyan átalakítók, amelyek az érzékelésre szánt jelet elektromos jellé alakítják [7].

Az ilyen átalakítók közé tartozik a mikrofon, a kamera, illetve a vevő oldalán a hangszóró és habár ez nem a legmodernebb megoldás, a katódsugárcső, ami sok monitor, vagy tévékészülék szerves részét alkotja. Már utaltam arra, hogy a zajos csatornán történő jeltovábbítás központi probléma ebben az ágazatban.

Ezeket a jeleket többféle szempontból lehet csoportosítani. A jel alapesetben lehet analóg, amely a függvény képét tekintve folytonos, vagy lehet digitális, amely diszkrét. Azonban létezik olyan jel is, ami csak időben diszkrét, ez a mintavételezett jel, valamint létezik kvantált jel is, ez esetben a jel csak amplitúdóban diszkrét.

Egy másik osztályozási szempont szerint a jelek lehetnek determinisztikusak, vagy sztochasztikusak. A determinisztikus jelek egyértelműen leírhatók időfüggvénnyel, amely alapján meghatározhatjuk a tulajdonságait. A sztochasztikus jeleket statikus jellemzőkkel írhatjuk le (átlagérték, várható érték stb). Ilyen esetben egyetlen időfüggvényből nem feltétlenül vonhatunk le végleges következtetéseket. A determinisztikus jelek esetében azonban igen, mivel a determinisztikus jel egyetlen konkrét függvény.

## Bluetooth

A Bluetooth egy távközlési szabvány, amely vezeték nélküli adatátvitelt tesz lehetővé. Tényleges fejlesztése 1998-ban kezdődött el, többek között az ERICSSON és a NOKIA közreműködésével [5]. Az Ericsson, az IBM, az Intel, a Nokia és a Toshiba vezetésével az üzleti érdekek harmonizálására létrehozták a Special Interest Group (SIG) nevű csoportosulást, amely első ténykedéseként megalkotta a Bluetooth szabványt. Mivel a szabvány szabadon felhasználható, a csoporthoz azóta közel ezer új tag csatlakozott, olyan szoftver- és hardvergyártók, akik rövidesen 'Bluetooth' termékek seregével kívánnak előrukkolni a piacon. Így egyebek mellett számos mobiltelefonba is beépítik ezt a technológiát.

A Bluetooth vezeték nélküli technológia a kábelek kiváltásánál jóval többet jelent: univerzális hidat alkot a jelenlegi adatátviteli hálózatok felé, periférikus interfészt biztosít és módszert, amellyel a helyhez kötött hálózati infrastruktúrán kívül kisebb ad-hoc magáncélú készülékcsoportokat alkothatunk. Gyors vételnyugtázást és frekvenciaugratásos kapcsolást használ, hogy még zajos rádiókörnyezetben is jó minőségű kapcsolatot biztosítson.

Rengeteg alkalmazása van számítógépek, mobiltelefonok, palmtopok és egyéb készülékek közötti kommunikációra, amit rádiókapcsolattal valósít meg. Jellemzően kis hatótávolságú. Az 1.2-es verzió 723 kb/s-os, a 2.0-s Bluetooth pedig 3Mb/s-os adatátviteli sebességet tesz lehetővé a világszerte elérhető 2,45 gigahertzes frekvencián. A hordozható eszközökön való elterjedését többek között az alacsony energiafogyasztása tette lehetővé.

A rádiós kapcsolat miatt a Bluetooth kommunikációt nem lehetetlenítik el a falak sem. A készülékek osztályuktól függően az alábbi távolságon belül képesek kommunikálni [6]:

| Osztály | Teljesítmény | Hatótáv |
|---|---|---|
| 1 | 100 mW (20 dBm) | 100 méter |
| 2 | 2,5 mW (4 dBm) | 10 méter |
| 3 | 1 mW (0 dBm) | 1 méter |

Működési tartománya a szabadon felhasználható, 2.4 GHz-es úgynevezett ISM rádió sáv. Megvalósítása az úgynevezett frekvencia ugrásos, szórt spektrumú technológiával történik. A frekvencia sávok ugró csatornákra vannak osztva, amelyeket a kapcsolat idején álvéletlen módszerrel választanak ki az eszközök.

A rendszer számára definiált úgynevezett *piconet* hálózatot egyszerre 8 eszköz használhatja a csatornákon megosztozva.

Vezérlőjelei a soros kommunikáció által használt vezérlőjelekhez hasonlíthatók. Tulajdonképpen vezeték nélküli módon soros kábelt emulál, ami növeli a kompatibilitást, mert rengeteg kommunikációs porton kommunikálni képes eszköz esetében tudjuk alkalmazni.


3.ábra. USB csatlakozású Bluethoot adapter

A vezeték nélküli rendszerekben az átvitt adatok biztonsága az egyik legfontosabb kérdés: vajon nem fér-e illetéktelen személy legfontosabb adatainkhoz, miközben azok látszólag védtelenül közlekednek a levegőben? A Bluetooth rendszerben a felhasználók védelmére három biztonsági szolgáltatást is szabványosítottak. A Hitelesítés egy kulcs vagy PIN kód alapján azonosítja a pico-hálózatba belépő egységet. A Meghatalmazással kizárhatunk egy adott eszközt a pico-hálózatból. A Titkosítással pedig mások számára hozzáférhetetlenné tehetjük a rádióhullámok hátán vitt adatainkat. A titkosítási kulcs hossza 40 vagy 64 bites lehet.

A Bluetooth, mint kommunikációs csatorna felhasználási módjairól, valamint az általa használt üzenetküldési –és fogadási módszerekről egy későbbi fejezetben lesz szó részletesen.

## Bluetooth kapcsolat felépítése

A kapcsolat felépülése előtt mindegyik Bluetooth egység (mobiltelefon, laptop stb) *standby* állapotban van. Ebben az állapotában mindegyik berendezés 1,28 másodpercenként "hallgat bele" 32 előre meghatározott csatornába, hogy megállapítsa: érkezik-e a számára *PAGING* (keresés) üzenet, vagy sem. A rendszer a *MAC* (Media Access Controll) cím alapján keres. A *MAC* hárombites cím, amely azonosítja az eszközt egy adott piconeten belül a kapcsolat idejére. Ha az egységnek valamilyen oknál fogva még nincs *MAC* címe, akkor előbb egy *INQUIRY* (lekérdezés, érdeklődés) üzenet érkezik a készülék felé, amelyet a különböző Bluetooth eszközök megkeresésére használnak. A felébredési idő, mialatt az egység felkészül az adatátvitelre maximálisan 2,6 másodperc, majd az adatátvitel után a készülék újra *standby* állapotba kerül.

A rendszer háromféle energiatakarékos üzemmódot is lehetővé tesz (*HOLD*, *PARK*, *SNIFF*). A szolga eszköz vagy automatikusan kapcsolódik takarékos üzemmódba, vagy a mester eszköztől kap parancsot arra. A *HOLD* üzemmód több összekapcsolt piconet esetén használatos, ekkor csak a *PAGING* üzenetekre figyel a készülék. A *SNIFF* állapot időtartama beállítható, ezalatt az egység "félálomban" figyeli a piconet adatforgalmát. A *PARK* módban a berendezés elveszti a *MAC* címét, és csak megfelelő időközönként figyeli a broadcast üzeneteket. A készülék nyilván a *PARK* üzemmódban bánik a legtakarékosabban az energiával, ekkor csak 0,03 mA az energiafogyasztása, *HOLD* esetén 0,06 mA, a *SNIFF* módban pedig 0,1 mA. Ezzel szemben az adatátvitel közben a fogyasztás 8-30 mA között változik.

# A Bluetooth alkalmazása

A Bluetooth először talán a PDA-k, mobiltelefonok terén terjedt el, ahol fájlok és adatok átvitelét tette és teszi lehetővé akár a mobil készülékek között, vagy a számítógépre történő archiválás során, kábelek használata nélkül. Nem feledkezhetünk meg azonban arról, hogy a Bluetooth kapcsolatnak kicsi a sávszélessége, és csak kis területen képes stabilan kommunikálni a vele kapcsolatban lévő eszközzel. Ezért habár számítógép és számítógép között is létre tudjuk hozni a kapcsolatot, valószínűleg nem ez lesz a leggyorsabb módja az adatátvitelnek.

Ugyanakkor a számítógépeknél is van sok kiváló alkalmazása ennek típusú kapcsolatnak. Nyomtatók, billentyűzetek, egerek és más perifériák csatlakoztatása során, ahol nem követelmény a nagy adatátviteli sebesség, ugyanakkor helyettesíti a vezetékeket, és így kényelmesebbé teszi a felhasználást.

A Bluetooth technológia alighanem forradalmasíthatja az értekezleteket. A résztvevők csak az asztalra helyezik a laptopjukat, és a rádióhullámok segítségével máris felépül a láthatatlan kapcsolat. Ha pedig valaki éppen egy e-mailt kapna, a zsebében lapuló mobil azt is azonnal a képernyőjére küldi az éter hullámain keresztül.

Visszatérve a mobiltelefonokhoz, az autóban való utazáskor a Bluetooth-on csatlakoztatott headsetek megoldást adnak a telefonálásra, miközben vezetünk.

Az én feladatomban a Bluetooth kommunikáció a számítógép és a robot között valósul meg olyan módon, hogy a számítógépen futó program vezérlőjeleket küld a robot moduljának.

# Robotika és a telekommunikácó együtt

Az olyan esetekben, amikor embernek nem lehetséges, vagy túl kockázatos egy helyen, vagy adott körülmények között tartózkodni. A robot információkat tud adni a veszélyes, vagy ember számára megközelíthetetlen helyről. Ebben az esetben távvezérlés valósul meg, a robotot emberek irányítják. A robotra szerelt kamerák és szenzorok segítségével egy megfelelően kiépített vezeték nélküli csatornán folyhat a kétirányú kommunikáció. A robotok megjelennek a katonaság, vagy az űrkutatás számos területein, de manapság már elengedhetetlen a gépesített munkavégzés a legtöbb gyártási folyamatnál.

A robotikában megjelenik a mesterséges intelligencia és a távvezérlés egyidejűleg. Ez azt jelenti, hogy a robot önállóan képes döntéseket hozni a rajta futó program alapján, és a szenzorai által gyűjtött információkat figyelembe véve, viszont távvezérléssel ezeket a döntéseket felül lehet bírálni. Fordítva is igaz. Lehetnek védelmi mechanizmusok a robotban, amelyek nem engedik, hogy egy figyelmetlen emberi (vagy gépi) vezérlés bajba sodorja a robotot.

A konkrét feladatomban is megjelenik ez a védelmi mechanizmus, ugyanis a robot mozgása során folyamatosan figyeli, hogy van –e előtte akadály egy meghatározott távolságon belül. Amennyiben az akadály a gép által tervezett útvonalra esik, a program felülbírálja az útvonalat, kikerüli az akadályt, majd az új pozícióból meghatározza az új útvonalat. Ennek az algoritmusnak a pontos folyamata egy későbbi fejezetben lesz részletezve.

# A Lego Mindstorms robot, NXT technológia

A Mindstorms a Lego által kifejlesztett modulárisan megépíthető robot, amelyet a gyártók instrukciója szerint megépíthetünk többek között kocsi, vagy emberalak formájában is. Ezen felül pedig bármit, amit össze tudunk rakni a lego elemekből. A csomagban szerepelnek a lego lakatrészek, az NXT modul, kézikönyv és telepítő CD. Praktikussága és sokoldalúsága miatt oktatási intézményekben és otthonokban egyaránt elterjedt.

Az NXT a MINDSTORMS robot agya. Ez egy intelligens, számítógéppel programozható modul. Az NXT-hez három motor csatlakoztatható, amely a robot egy részegységének, vagy egészének precíz mozgatására alkalmas.

Az alap kiszerelésben négy szenzor csatlakoztatható az agyhoz, ezek [3]:

1. TouchSensor – egy nyomógomb segítségével tud érzékelni és reagálni.
2. SoundSensor – segítségével hang-hatásokra tehetjük reakcióképessé robotunkat.
3. LightSensor – különböző fényerősségeket és színeket képes megkülönböztetni.
4. UltrasonicSensor – a szenzor előtt levő tárgytól való távolságot képes megállapítani.

Az NXT rendelkezik hangszóróval, grafikus kijelzővel, és nyomógombokkal. A számítógépen megírt programot USB kábelen keresztül, vagy már a távvezérléshez is használható Bluetooth csatornán keresztül lehet le- és feltölteni.

## Az NXT Bluetooth – protokollja

Master – Slave kapcsolat valósul meg, a Master összesen három szolga egységgel tud kapcsolatot tartani egyidejűleg. A firmware frissítést kivéve minden funkció elérhető ilyen kapcsolattal, akárcsak USB kábelen, viszont az USB host alapú protokoll, így több NXT között nem lehet megvalósítani vele a kommunikációt, Bluetooth segítségével viszont igen!

A Master oldal mindig az, amely a Bluetooth kapcsolatot létrehozta. Ha az NXT modul Bluetooth menüjéből hoztuk létre a kapcsolatot, akkor a robotunk a Master, ha pedig más eszköz hozta létre a kapcsolatot, akkor az NXT egyszerűen csak fogadja a csatlakozási kérelmet Slave-ként.

## Az üzenetváltásról

Amikor a Master üzenetet küld a Slave-nek, egy visszajelzést vár a Slave-től, vagy arra számít, hogy a Slave visszajelzés nélkül fogadja az adatokat [2]. Abban az esetben, amikor a Master adatokat kér a Slave oldaltól, a visszajelzés tartalmazza a kért adatot, amennyiben ez az adat elérhető. Ha ez az adat nem elérhető, vagy nem fut program a Slave-en, akkor egy hibakódot küld a Masternek. A kapcsolat *tétlensége (idle)* esetén a következő esemény biztos, hogy egy a Mastertől érkező üzenet lesz. Ez az üzenet pedig meghatározza, hogy szükség van-e válasz üzenetre, vagy sem, és ebből az információból mindkét félnek ismét egyértelművé válik, hogy ki fog legközelebb üzenni. Egy ilyen üzenet továbbítási procedúra *(transmit)* lezajlása után a kapcsolat ismét *tétlenné* válik, és a folyamat elölről kezdődik.

Amikor egy program üzenetet küld, vagy fogad, mindig meghatároz egy postaládát *(mailbox)* amely 0 és 9 közötti sorszámmal rendelkezhet (az NXT-G nyelvben ez 1 - 10 sorszámok). Ezzel 10 virtuális csatorna nyílik a Slave számára a kommunikációhoz. Ahogy azt már említettem, a Master egyidejűleg 3 Slave-hez képes csatlakozni, ezért amikor a Master elküld egy üzenetet, a program határozza meg, hogy melyik Slave és melyik postaláda legyen a címzett. A Slave oldalaknak egyszerűbb dolguk van, ugyanis a Masterhez csak egy postaláda tartozik, és a Master csak akkor tudja, hogy melyik Slave-től érkezett üzenet, ha ezt

az információt az üzenet tartalmazza.

Az NXT esetében a kommunikáció négy rendszerhívásból tevődik össze. Ezek közül kettőt csak a Master használ.

Az első ellenőrzi a Bluetooth rádió foglaltságát *(NXTCommBTCheckStatus)*. Amennyiben foglalt, a programnak mindaddig várakoznia kell, amíg a rádió nem szűnik meg foglalt jelzéseket adni.

A második rendszerhívás *(NXTCommBTWrite)* üzenetet kézbesít, ez csak akkor hívódik meg, amikor a rádió nem elfoglalt.

A harmadik rendszerhívás *(NXTMessageRead)* az üzenetfogadásért felel, a Master oldal határozza meg, hogy melyik postaládából történjen az olvasás. Ugyanez a rendszerhívás lép életbe, amikor a program kér üzenetet a Mastertől.

A negyedik rendszerhívást *(NXTMessageWrite)* csak a Slave oldal használja. Ennek segítségével üzenetet tud küldeni a Masternek.

Sajnos előfordul, hogy az NXT Bluetooth protokoll elveszít üzeneteket. Ez a postaládákhoz tartozó sorbanállás *(queue)* korlátoltsága miatt van, ugyanis minden egyes postaláda mindösszesen 5 üzenetet képes tartalmazni. Amikor egy üzenetnek mindenképp be kell kerülnie egy megtelt sorba, akkor a legrégebbi üzenet törlődik, elvész. Viszont csak a Slave oldalról küldött üzeneteknek vannak sorbanállásra kényszerítve, ugyanis a Master azonnal elküldi az üzeneteket, és csak aszerint fogad üzenetet, ahogy az eseménykezelő rendszer meghatározza *(poll-interval)*. Tehát a Masteren futó program kéri az üzenetet.

Más is okozhatja üzenetek elvesztését. Ha a Master hamarabb küld üzenetet, mint ahogy a Slave oldalon elindulna a program, szintén elvésznek az üzenetek. Másfelől pedig, ha a Slave üzenetküldése során a Slave-en futó program hamarabb befejeződik, mint ahogy a Master elkérte volna a sorból az üzeneteket, az üzenetek elvesznek, mivel a program leállásakor a sor törlődik.

Tudunk azonban lépéseket tenni annak érdekében, hogy elkerüljük a sor túlcsordulást és az üzenetek elvesztését. Ezen módszerek közül nem mindent lehet megvalósítani az NXT oldalon, viszont a PC oldalon igen!

A legegyszerűbb módja a túlcsordulás elkerüléséhez, ha üzenetküldés előtt ellenőrizzük a sor állapotát, és csak akkor küldünk, ha nincs telítve.

Ugyan az NXT memóriája korlátolt, de a PC memóriáját felhasználva ki is bővíthetjük a sort, ezzel több férőhelyet biztosítva az érkező üzenetek számára.

Arra is van mód, hogy a Master ellenőrizze, hogy fut-e program a Slave oldalon, és így az ezzel kapcsolatos üzenetvesztéseket is kiküszöböljük. Ezt úgy valósíthatjuk meg, hogy az algoritmus a Slave-től való üzenetfogadással kezdődik a Master oldalon. Amíg ez nem történik meg, addig további üzenetküldések sem történnek, így elkerüljük az üzenetvesztést.

Ezekkel és ezekhez hasonló módszerekkel a nem-megbízható kapcsolatot megbízhatóvá tudjuk átalakítani.

Most pedig hasonlítsunk össze a korábban említett nyelvek közül két különböző típusút a Bluetooth kommunikáció megvalósítása tekintetében.

- NXT-G
    - 10 csatornán képes egymástól független üzeneteket küldeni
    - Fél-duplex (13 tranzakció / sec)
    - Van üzenetküldési várakozási idő, a slave figyeli a Master-t, hogy mikor küldhet adatot.

- RobotC
    - Master-Slave kapcsolatra optimalizált, de támogatja a Multi-Slave kapcsolatot is
    - Nincs üzenetküldési várakozási idő, a slave közvetlenül tud üzenetet küldeni a Masternek
    - Full-duplex (36 tranzakció / sec)

## Vezérlés mobiltelefonnal

Van rá mód, hogy Bluetooth kapcsolatot hozzunk létre az NXT és egy mobiltelefon között, viszont a mobiltelefonnak támogatnia kell a Java alkalmazásokat, ugyanis az NXT-t vezérlő programnak szüksége van a Java támogatásra. A Lego NXT Mobile Application nevű szoftverével vezérelhetjük a robotunkat, vagy futtathatjuk a modulon levő programokat.

A Lego NXT Mobile Application jelenleg a következő készülékeket támogatja:

**Nokia:** 6680, 3230
**Sony Ericson:** W800i, W550i, K610i, K800i, K750i, Z710i, Z550i, K510i
**BenQ-Siemens:** CX75, X75

4. ábra. NXT vezérlése mobiltelefonról

# Az NXT programozása

A programunkat számos nyelvben megírhatjuk. Ezek közül néhány:

**NXT-G:**

- Grafikus nyelv
- Windows, Mac OSX
- az alapcsomag tartalmazza

Ezt a nyelvet a gyerekek számára is ajánlják, ugyanis nagyon egyszerű kezelni. Különböző "dobozokat" lehet letenni (pl. motor, szenzorok), és azokat összekötni. Támogatja a Bluetooth kommunikációt. Gyakorlatilag kódírás nélkül tudunk programot készíteni. Ahogy az IDE-ben található példaprogramból is látszik, meg lehet valósítani vele precizitást igénylő feladatokat is, azonban egy kódíráshoz hozzászokott programozónak valószínüleg nem ez a nyelv lesz a legjobb választás.

**RoboLab:**

- Grafikus nyelv
- Windows, Mac OSX

A RoboLab-ot szintén használják gyerekek oktatására. Az NXT-G-hez hasonlóan egyszerű kezelni, viszont egyáltalán nem támogatja a Bluetooth kommunikációt.

**NI LabVIEW Toolkit:**

- Grafikus nyelv
- Windows, Mac OSX

Ez egy kiegészítés az NI LabVIEW adatfolyam programozási nyelvhez. Ez is grafikus, de sokkal magasabb szintű nyelv, mint az előző kettő. Lényegesen több lehetőséget kínál az alapprogram eszközeivel együtt. Ez a toolkit az NXT-vel kapcsolatos eszközökkel bővíti a "VI" palettát (motor, szenzorok, Bluetooth mailbox, stb).

**RobotC:**

- C nyelv
- Windows

A RobotC rendelkezik saját fejlesztői környezettel. A saját firmware-e eltér a gyáritól (NXT-G), ezért az első lépés a firmware frissítés. Ezt egyszerűen a menüsorból a *Firmware update* – re kattintva megtehetjük. A második lépés a programírás előtt a motorok és szenzorok beállítása. Ez egy rövid kódot fog generálni, ami később segítségünkre lesz a perifériákra történő hivatkozáskor. Egyértelmű és szimpatikus IDE, egy C programozónak sem okozhat problémát az ebben a környezetben történő fejlesztés. Később kitérek a konkrét Bluetooth-os függvényekre is.

**Mindsqualls:**

- C# nyelv
- Windows

Ez egy .NET-es programkönyvtár. A fejlesztéshez és az ebben megírt alkalmazások használatához szükséges a .NET framework. Könnyen használható elemeket tartalmaz, mindösszesen egy DLL fájlra van szükségünk, hogy használhassuk az ebben megírt osztályokat. A Bluetooth kommunikációval kapcsolatos utasításokra egy későbbi fejezetben térek ki.

**leJOS NXJ:**

- Java nyelv
- Windows, Linux, Mac OSX

Nem tartalmaz önálló fejlesztői környezetet. Ez gyakorlatilag egy *plugin* a már minden java programozó által ismert Netbeans és Eclipse IDE-khez. Ingyenesen letölthető és támogatja a Bluetooth kommunikáció minden formáját. Firmware frissítésre itt is szükség van, erre például lehetőséget ad az Eclipse Plugin for leJOS NXJ 0.5. De van más út is a frissítéshez, egy parancssor ablakban az *nxjflash* parancs kiadása után a firmware frissül, ezt az NXT kijelzőjén megjelenő logo is jelzi, valamint a menürendszer módosulása.

# Robotvezérlő utasítások

## A Mindsqualls programkönyvtár

Az általam létrehozott projektben a Mindsqualls programkönyvtár döntő szerepet játszik a vezetéknélküli kommunikáció megvalósítása során. Azért választottam ezt a programkönyvtárat, mert egyszerű a sorosport-kezelése, a grafikus felhasználói felület tervezése, valamint a hálózat kezelése. Ez egy C# nyelvű .NET könyvtár, amely Bluetooth kapcsolaton keresztül teszi lehetővé az NXT távvezérlését.

```
// NXT brick deklarálása a portszám megadásával.
NxtBrick brick = new NxtBrick(40);

// Motorok csatlakoztatása az NXT B és C portjára.
brick.MotorB = new NxtMotor();
brick.MotorC = new NxtMotor();

// A motorok szinkronizálása, a biztos együtthaladás
// érdekében.
NxtMotorSync motorPair = new NxtMotorSync(brick.MotorB,
brick.MotorC);

// NXT csatlakoztatása.
brick.Connect();

// Motorpár forgatása (75% power, 3600 degree).
motorPair.Run(75, 3600, 0);
Console.WriteLine("Running...");

// Kapcsolatbontás.
brick.Disconnect();
```

## NXT Bluetooth elérés RobotC-ben

A RobotC egy C nyelvű, saját fejlesztői környezettel rendelkező szoftver. Miután frissítettük a saját firmware-ével az NXT modulunkat, lehetőségünk van olyan alkalmazásokat írni, amelyekben a bluetooth kommunikáció minden lehetséges irányban működőképes (PC és NXT között, NXT és NXT között, valamint NXT és egyéb eszköz között). A fejlesztés közben egy fejlett debug rendszer segítségével tudjuk nyomonkövetni a program futását. Sokak számára viszont nagy hiányossága az IDE-nek, hogy csak Windows-on képes futni.

```
//csatlakozás portszám és név megadásával.
btConnect(nResult, nPort, sFriendlyName);

//kapcsolat bontása adott portról.
btDisconnect(nResult, nPort);

//az összes kapcsolat bontása.
btDisconnectAll(nResult);

//eszköz eltávolítása.
btRemoveDevice(nResult, sFriendlyName);

//keresés indítása.
btSearch(nResult);

//keresés leállítása.
btStopSearch(nResult);

//Bluetooth kikapcsolása.
setBluetoothOff(nResult);

//Bluetooth bekapcsolása.
setBluetoothOn(nResult);
```

```
//láthatóság beállítása.
setBluetoothVisibility(nResult, bBluetoothVisible);

//beállítja az alapértelmezett PINkódot.
setDefaultPIN(nResult, sPIN);

//eszköznév beállítása
setFriendlyName(nResult, sFriendlyName);

//fájlküldés.
transferFile(nResult, nPort, sFileName);

//visszaadja az aktuális Bluetooth állapotot.
nBluetoothCmdStatus

//visszaadja a legutóbbi Bluetooth parancsot.
nLastBTCommand
```

## NXC Bluetooth programkönyvtár

Az NXC viszont fut minden rendszeren, Windows-on és Unix alapú rendszereken egyaránt. A saját fejlesztői környezetében „C-szerű” nyelven programozhatunk. Hátránya a RobotC-hez képest, hogy nem támogatja a más eszközökkel való kommunikációt Bluetooth csatornán keresztül.

```
//A használathoz szükséges header-állományok
#include "NXCDefs.h"
#include "BTlib.nxc"
```

```
//NXT csatlakozásának ellenőrzése
BTCommCheck(CONN);

//BT tranzakció befejezésére való várakozás
BTWait(CONN);

//String küldése (számokat stringként küldünk, a
//konvertáláshoz NumToStr(n) függvényt használjuk
BTSendMessage(CONN,MAILBOX,MSG);

//String üzenet fogadása. Az üzenet az MSG //változóba kerül
BTReceiveMessage(CONN,MAILBOX,FLUSH);
```

# A navigációs feladat megvalósítása

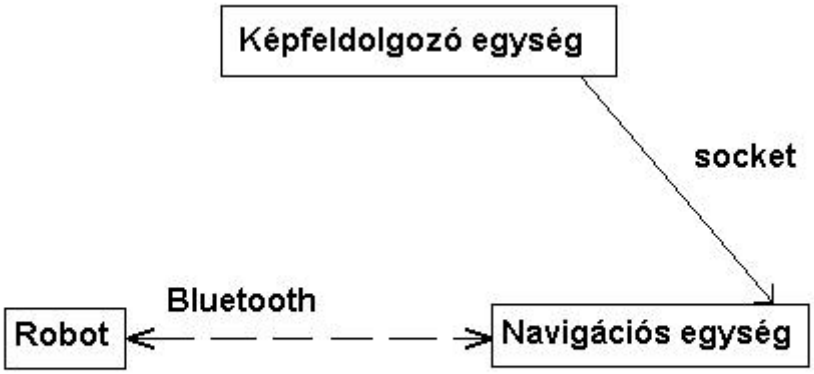
Képzeljünk el egy olyan szituációt, amikor a robotunkat olyan környezetben kell távolról irányítanunk, ahol a robotot körülvevő akadályok között kell navigálni. A fedélzeti kamera valószínűleg nem teszi lehetővé, hogy lássuk a robot közvetlen környezetét minden irány felől, ha pedig netalán veszélyes helyen kell a robotnak közlekednie, nem bízhatunk meg csupán a különböző távolságmérő szenzorokban, szükséges az emberi irányítás. Ilyen esetben jó megoldás lehet, ha a robotot felülről látjuk, például egy műhold kamerája segítségével.

Egy másik esetben, ha az emberi felügyelet nem indokolt, viszont a legtöbb információt továbbra is egy felülről néző kamera tudja szolgáltatni, van lehetőség a navigálás automatizálására is.

A feladatomban ez a navigálási módszer is szerepet játszik, csak kisebb területen van megvalósítva. A feladat tehát, hogy a robot egy felülre felszerelt kamera által belátott terület egyik pontjából eljusson egy másik pontba emberi beavatkozás nélkül, és a kocsi abban a pozícióban legyen a végállapotban, amelybe az első iránybaállításkor kerül.

A kamera folyamatosan információt szolgáltat a robot pályán való elhelyezkedéséről, és a robotkocsi pozíciójáról is. A kocsi tetejére egy kétszínű papírlap van ragasztva, így a program a színek egymáshoz való elhelyezkedése alapján tudja a kocsi pozícióját, tehát hogy merre néz a kocsi eleje. A programnak meg kell határozni az útvonalat az aktuális tartózkodási helytől a célhelyig, majd a már előzőleg kiépített bluetooth kapcsolaton keresztül vezérlőjeleket kell küldenie a motoroknak, hogy a meghatározott útvonalnak megfelelően irányítsa és hozza mozgásba a kocsit. A kocsi mozgása közben a kamera folyamatosan aktuális információkat szolgáltat a programnak, amely ezeket kiértékeli, és szükség esetén korrigálja a kocsi mozgásának irányát, vagy sebességét. Amikor a program a kamerától kapott információt úgy értékeli ki, hogy a kocsi elhelyezkedése megegyezik a célhellyel, és a kocsi pozíciója is megfelelő, akkor a program befejezi működését.

## A rendszer áttekintése




A Képfeldolgozó egység egy PC –ből és egy hozzá csatlakoztatott kamerából áll, amely felülről látja be a mozgásteret. A kamera függőlegesen lefelé néz. A PC –n egy Ubuntu Linux van telepítve, és azon fut a Mezzanine képfelismerő program és egy segédprogram.

A Robot nevű egység a már korábban tárgyalt Lego készletből összeszerelt háromkerekű robotkocsiból és a rá erősített piros és zöld színű kartonlapból áll.

A Navigációs egység tartalmaz egy PC –t, amelyen Windows Xp rendszer fut, amelyre telepítve van a .NET framework 3.0, továbbá a Visual Studio Express 2008. Az egységnek a Bluetooth kommunikációt lehetővé tevő Bluetooth adapter is része.

A Robot egység a navigációs egységgel a Bluetooth kapcsolaton keresztül kommunikál. Mivel egyrészt a robotnak a PC utasításokat küld, másrészt a robottól a PC szenzoradatokat olvas, a kommunikáció kétirányú. A Képfeldolgozó egység a Navigációs egységnek koordináta adatokat küld, köztük a kommunikáció egyirányú.

# A robot fizikai felépítése

A robotnak két meghajtott kereke van, és egy harmadik támasztókerék. Ez azért előnyös, mert így nincs szükséges differenciálmű beépítésére. Ha a kerekek ellentétes irányba forognak, a robot gyakorlatilag egyhelyben képes megfordulni. Egy ultrahangos távolságmérő szenzor van a robotra szerelve az akadályok érzékelhetősége végett. A robot nem tudja mérni a saját elmozdulását és forgását. A robot azért készült a Lego készletből, mert egyszerűen és gyorsan összeszerelhető, és a konkrét feladatomnál nem szükséges, hogy a robot erős, vagy gyors legyen, ugyanakkor kellőképpen pontosan mozgatható.

# A Képfeldolgozó egység

## Mezzanine

A Mezzanine egy nyílt forráskódú programcsomag, amely mozgásban levő tárgyak követését teszi lehetővé két dimenzióban [4]. Ennek megvalósítására kamerát használ, amellyel egy színjelölésekkel ellátott objektumnak képes meghatározni a pozícióját, valamint a kamera által belátott területen való elhelyezkedését, koordinátáit. A szoftver nagy előnye, hogy bármilyen színes kamerával képes működni, még a legolcsóbbakkal is, így nem kell méregdrága kamerát beszerezni a feladat elvégzéséhez, ugyanis a program a kamera lencséitől származó torzításokat kikorrigálja. A programcsomagot C nyelven fejlesztették Linux rendszeren. A Mezzanine tulajdonképpen egy serverprogram, az aktuális koordináták és pozíció közlése az IPC library segítségével valósul meg.

A Mezzanine-nak három fő része van:

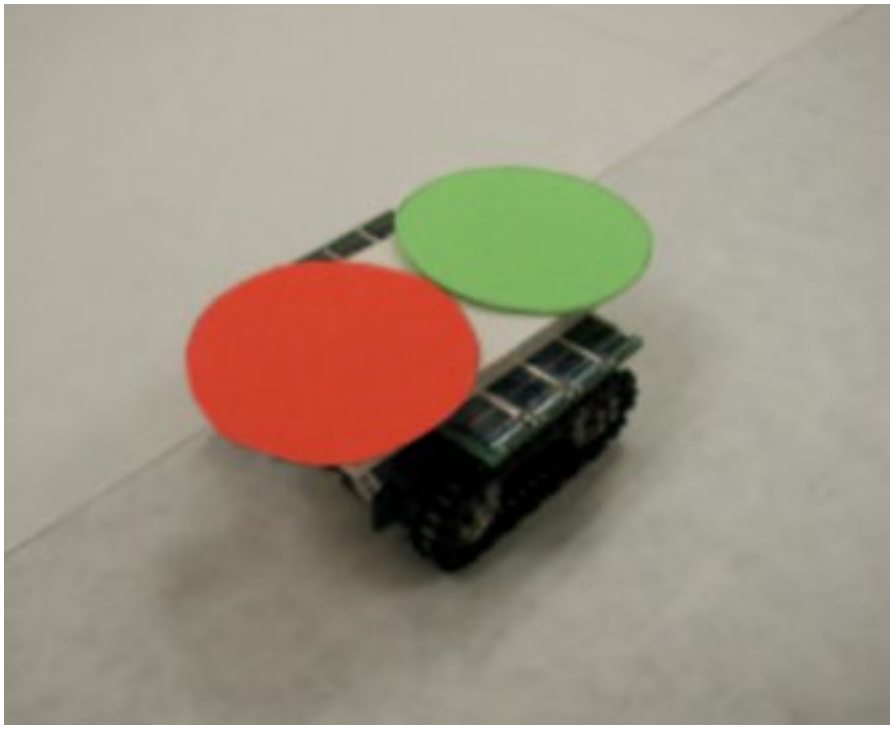
- Mezzanine: a követő program
- Mezzcal: a futtatás előtt kalibrálásra van szükség, a színek beállítására, ezt követően a későbbiekben a Mezzanine tudja mihez viszonyítani a pozíciót és a koordinátákat

- Libmezz: egy IPC[1] könyvtár a mezzanine-nal való kommunikáláshoz

A Mezzanine Video4Linuxot[2] használva képkockákat vételez a kamera segítségével bizonyos időközönként, és ezekből a képekből meghatározza a földön, vagy bizonyos mértékig a föld felett levő objektum koordinátáit és pozícióját. A PC –re csatlakoztatott webkamera képe a /dev/video0 állományon keresztül érhető el, és a Mezzanine is éppen itt keresi az alapértelmezések szerint.

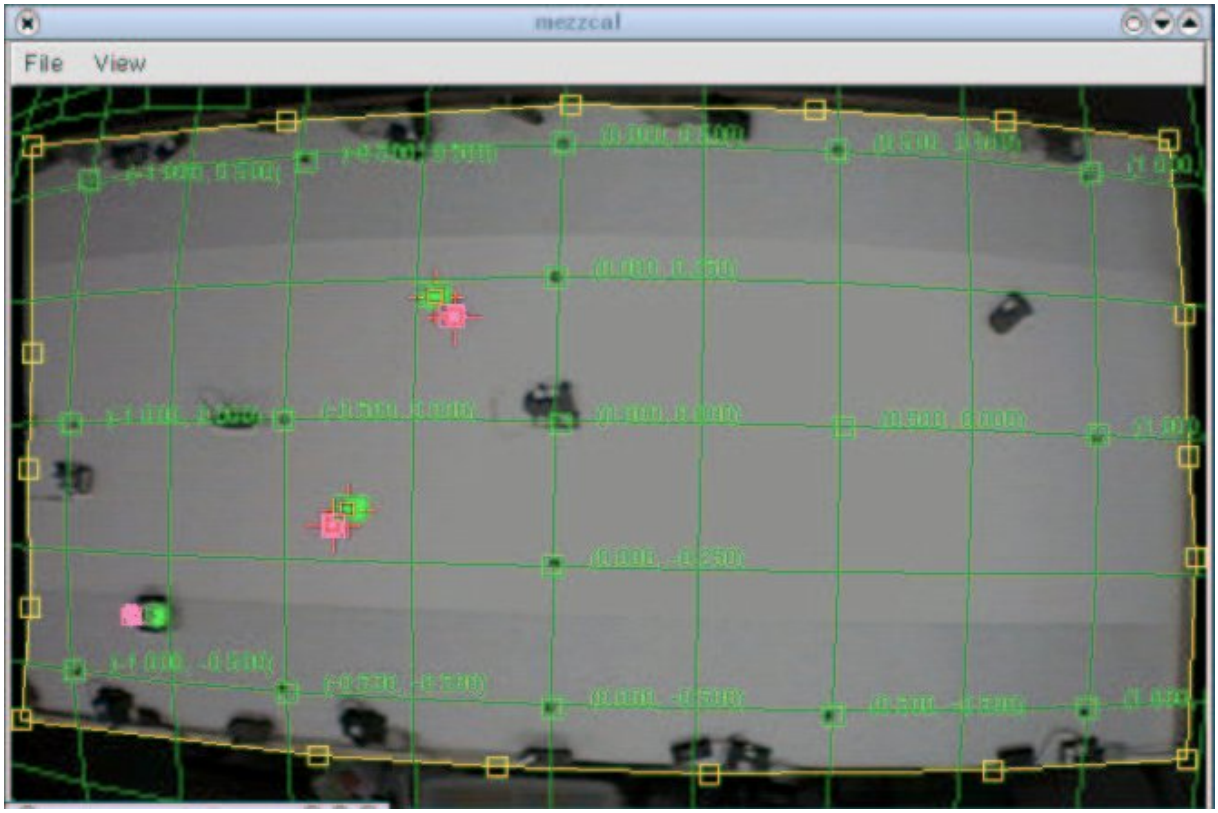
Egy PIII 700Mhz –es számítógépen 30 képkockát képes feldolgozni másodpercenként, mialatt a processzort 70% -on veszi igénybe.

A Mezzcal egy grafikus felhasználói környezetet biztosít a rendszer bekalibrálásra. Többek között azokat a színeket határozhatjuk meg, amelyekkel megjelöltük a követni kívánt objektumot. A konkrét feladatom elvégzése során a piros és zöld színek bizonyultak legmegfelelőbbnek.



5. ábra. Színekkel megjelölt robotkocsi

[1] http://en.wikipedia.org/wiki/Interprocess_communication
[2] http://en.wikipedia.org/wiki/Video4Linux




6. ábra. Kalibrációs rács

A Libmezz library egy egyszerű C interfészt nyújt, amely lehetővé teszi a parancsok küldését a Mezzanine-nak, valamint adatokat tud olvasni onnan. Új elérhető adat esetén Unix signalokat használ az értesítéshez. Minden egyes új kép feldolgozása után a Mezzanine egy SIGUSER1 jelet küld minden bejegyzett folyamatnak. A folyamatoknak erre a jelre az új adat kiolvasásával kell reagálniuk. A libmezz-t használó folyamatok automatikusan bejegyzésre kerülnek a library inicializálása során és használhatják a mezz_wait_event() függvényt, és így várakozhatnak az új adat átvételére.

A program működéséhez szükség van tehát Video4Linux –ra, amely egy videotámogatás Linuxhoz. Szükség van GTK+ 1.2 –re, ezt minden Linux disztribúcióhoz megtaláljuk, sőt a legtöbb más Unix alapú rendszerekhez is. Végül szükség van a GSL legalább 0.9 –es verziójára, amit szintén minden disztribúció csomagkezelőjében megtalálunk, habár gyakran nem része az alaprendszernek alapértelmezésben.

Mivel a Mezzanine Linux rendszeren fut, és a Mindsqualls programkönyvtárat használó navigátor program csak Windows rendszeren működőképes, ezért két számítógépet kell üzemeltetni a feladat elvégzéséhez. Kell tehát egy összekötőelem (Coord_server), amely megszerzi a robot koordinátáira és pozíciójára vonatkozó adatokat a Mezzanine-tól (az IPC library segítségével), majd azokat a hálózaton továbbítja a másik számítógépen futó navigátor programnak.

## Koordináta-szerver

Ezt a programot C nyelven írtam. Socketen keresztül továbbítja a Mezzanine-tól megszerzett adatokat a 20000-es porton. A Mezzanine programcsomag tartalmaz egy mintaprogramot, amely hozzáfér a /dev/video0 –on a koordináta adatokhoz, és a standard kimenetre írja azokat. Ezt egészítettem ki a hálózati résszel, ami lehetővé teszi, hogy a program szerverként működjön, és a koordináta adatokat ne a kimenetre írja, hanem a socketbe. Így a navigátor programnak már csak csatlakoznia kell ehhez a szerverhez, és már rendelkezésére is állnak az aktuális koordináta adatok.

Most nézzünk pár kódrészletet a socketprogramozással kapcsolatban.

```
// socket készítése
int mySocket = socket(AF_INET, SOCK_STREAM, 0);

// A socket ellenőrzése
if (mySocket == -1) {
printf("Error Opening Socket!\n");
return -1;
}
// cím struktúra
struct sockaddr_in server;
```

```
// A cím struktúra megfelelő adattal való feltöltése
server.sin_family = AF_INET;
server.sin_port = htons(port);
server.sin_addr.s_addr = INADDR_ANY;

setsockopt(mySocket, SOL_SOCKET, SO_REUSEADDR, (char *)&on,
sizeof on);
setsockopt(mySocket, SOL_SOCKET, SO_KEEPALIVE, (char *)&on,
sizeof on);

// összerendeljük a socketet a szerverrel
if (bind(mySocket, (SOCKADDR*)&server, sizeof(server)) == -1)
{
printf("Bind Failed!\n");
closesocket(mySocket);
return -1;
}

// A server socket figyelni kezd

     if (listen(mySocket, 5) == -1) {
     printf("Listen Failed!\n");
     closesocket(mySocket);
     return -1;
}
```

# A Navigátor program

A navigátor programot tehát már a Windows rendszert futtató számítógépen használjuk. A 20000 –es porton csatlakozik a Coord-server programhoz, és átveszi az adatokat, amelyek eredetileg a Mezzanine –tól származtak.

Ezek feldolgozása után a program meghatározza a megadott célhelyig vezető útvonalat, majd a bluetooth csatornán keresztül elküldi az ennek megfelelő vezérlőjeleket a robotnak.

A robot navigálása során gyakorlatilag szabályzás valósul meg, a kamerán keresztül visszacsatolás történik. Erre azért van szükség, mert ugyan a Mezzanine pontos koordinátákat és pozíciót szolgáltat, valamint a navigátor program ezek alapján pontosan meghatározza az elfordulási szöget és útvonalat, a gyakorlatban a kocsi elforgatása mégsem mindig precíz különböző mechanikai és kerék tapadási tényezők miatt. Nem beszélve az esetleges akadály-elkerülési manőverekről, amelyek után mindig újra meg kell határozni a célhelyig vezető útvonalat, és ehhez folyamatosan aktuális adatokra van szükség.

A robot motorjait célszerű volt alacsony fordulatszámon használni, hogy ne legyen túl gyors a kocsi mozgása. Ez azért lényeges, mert ugyan a Mezzanine megfelelően kis időközönként képes képkockákat feldolgozni (30 képkocka másodpercenként), de mivel a navigáló program részéről az adatok átvétele hálózaton át történik, ez behozott némi lassulást, és így kevésbé volt folyamatos és részletes a robot aktuális koordinátáiról való értesülés. A kocsi sebességét úgy érdemes meghatározni, hogy a Mezzanine-tól kapott koordináta információk ne érkezzenek ritkábban, mint amilyen gyakorisággal a kocsi mozgása közben eléri a pálya azon pontjait, ahol már újabb aktuális információra lenne szüksége. Magyarán a robotnak ne kelljen tétlenül várnia a vezérlőjelekre, hanem lehessen a mozgása folyamatos.

A Mindsqualls programkönyvtár által felajánlott Bluetooth lehetőségekről, vezérlőjelekről és konkrét utasításokról már volt szó, most nézzük részletesebben a socket-programozást C# -ban, amely legalább olyan fontos részét képezi a programnak, mint a vezérlőjelek Bluetooth csatornán való küldése.

A sockethez szükséges osztályokat a System.Net.Sockets –ben érjük el. A csatlakozáshoz deklaráljunk TCP klienst, valamint a példányosításkor adjuk meg a címet és a portot.

```
TcpClient myclient;          //TCP kliens deklarálása
private NetworkStream networkStream = null;  // hálózati adatfolyam
private BinaryReader streamReader;
private BinaryWriter streamWriter;
private void csatlakozas()
{
try   //kivételkezelés
{
myclient = new TcpClient("localhost", 20000); //TCP kliens példányosítása
}
catch
{
Console.WriteLine("Nem tudtam csatlakozni a 20000 porton");
return;            // a program hiba esetén hibaüzenettel leáll
}
networkStream = myclient.GetStream();
streamReader = new BinaryReader(networkStream);    //olvasás a socketből
streamWriter = new BinaryWriter(networkStream);    //írás a socketbe
}
```

A socketbe byte-okat írhatunk és olvashatunk.

```
Byte bejovo = streamReader.ReadByte();
streamWriter.Write(byte);
```

Az x,y koordinátákat és a szögértéket String formátumban küldi a Coord-server, amelyet byte-onként ovlasunk be, majd a szókoz karakterek szerint három felé szedjük szét

(Split), és az így kapott három részt pedig Double típusúvá alakítjuk (Parse). Az aktuális koordinátákat az aktKoord objektumban tároljuk.
A Koord osztály (amelyből az aktKoord objektum létrejött) három double mezőből áll (x,y és angle).

A program leglényegesebb metódusa a *navigal* eljárás, amely a célhely eléréséig sokszor lefut. Az eljárás elején a program megvizsgálja, hogy elértük –e a célhelyet, majd meghatározza a célhelytől való távolságot (dist). Ezt követően a megfelelő szögértéket számolja ki arkusz tangens függvény segítségével. A következő feltételnek csak az első meghíváskor szabad lefutnia, ezért rögtön hamis értéket kap a *navkezdete* változó, amelyet globálisan deklaráltam. Itt megjegyezzük az kezdőpontból a célpontba mutató vektor irányát, ami azért fontos, mert a navigáció befejeztével ugyanebbe az irányba kell majd forgatni a robotot. Ezt követően a program megvizsgálja, hogy a célhelyen van –e a robot, valamint hogy a pozíciója megegyezik –e az előbb említett első pozícióval. Ez esetben befejezzük a navigálást. Ha pedig rossz irányban áll, akkor meghívja a *forgat* függvényt, átadva a kiszámított szögértéket. A folyamatos szabályzás azért is lényeges, mert a *forgat* függvény alkalmazása során az egyes elforgatások nem feltétlenül precízek, a már korábban említett mechanikai és tapadási problémák miatt. Ennek következtében a kocsi útja nem egyenes lesz, hanem annyi kis irányváltást fog tartalmazni, ahányszor meghatározza a program az aktuális pozícióból számított célhelyre mutató vektor irányát. Az irányváltások miatt viszont érdemes úgy meghatározni a robot előrehaladásának úrhosszát, hogy az minél rövidebb legyen, hogy ne legyen cikkcakkos a kocsi útja. Másfelől a pályán akadályok is előfordulhatnak, amelyek kikerülése során a robot eltér a tervezett útvonaltól, így a kikerülési manővert követően ismét meg kell határozni a megfelelő útvonalat.

Az akadályok észrevételéért egy ultrasonic szenzor a felelős, amelytől egy eseménykezelő rendszeren keresztül fogadja az adatokat a program. Amennyiben 15cm -en belül akadály van a robot előtt, felülbírálja a kiszámított útvonaltervet, elforgatja a kocsit 90 fokkal, majd halad a módosított irányban körülbelül 10 cm –t. Mivel a kiszámított útvonalon is csak körülbelül 10 cm –t halad előre a robot két útvonal ellenőrzés között, és a távolságszenzor pedig 15 cm távolságon belül vizsgálja az akadályt, így elkerüljük az akadálynak való ütközést.

A kódban szereplő *xyerr*, és *aerr* változók hibatűrési értékeket tartalmaznak. Minél kisebbre állítjuk ezeket, annál nagyobb pontosságot követelünk meg. Ezeket az értékeket a robot precizitásának és a terep minőségének függvényében érdemes beállítani. Ha túl kicsire állítjuk ezeket az értékeket, akkor a navigálás esetleg sohasem ér véget.

Alapállapotban a már említett *navkezdete* logikai érték igazra van állítva, majd addig hívom meg a *navigal* eljárást egy *timer*ből, amíg hamis értéke nem lesz.

```
public void navigal()
{
double dx = celx-aktKoord.x;
double dy = cely-aktKoord.y;
bool celonvan = Math.Abs(dx) < xyerr && Math.Abs(dy) < xyerr;

double dist = Math.Pow(dx*dx+dy*dy,0.5);

double joirany = Math.Atan2(cely-aktKoord.y,celx-aktKoord.x);

if (navkezdete)
{
     elsoirany = joirany;
     navkezdete = false;
}

if (celonvan)
{
     joirany = elsoirany;
     if (Math.Abs(elsoirany-aktKoord.angle) < aerr)
     {
          navigalas = false;
          return;
      }
}
```

```
if (Math.Abs(joirany-aktKoord.angle) > aerr) // ha rossz
irányba áll
{
     forgat(joirany-aktKoord.angle);
}
else
{
     if (sonarflag){
          forgat(Math.Pi / 2);
          motorPair.Run(50, 3600, 0);
     }else{
          motorPair.Run(50, 3600, 0);
     }
}
}
```

# Összefoglalás

A dolgozatban a Lego Mindstorms robotot és főként a hozzá tartozó NXT modult mutattam be, leginkább a Bluetooth által nyújtott lehetőségekre koncentrálva. Megemlítésre kerültek a megvalósításhoz használt programnyelvek és más programnyelvek lehetőségei, valamint korlátai. Első lépésben a Bluetooth kommunikáción alapuló távvezérlést tanulmányoztam és valósítottam meg a Mindsqualls programkönyvtár segítségével, majd egy webkamera és a Mezzanine program segítségével a feladat kibővült egy emberi beavatkozást nem igénylő alkalmazássá, amely a robotot egy adott területen a kiindulási pontból egy célpontba navigálja el. A robotra szerelt ultrasonic (távolságérzékelő) szenzor segítségével megvalósítottam, hogy a robot ne ütközzön neki előtte levő akadálynak, hanem amennyiben az akadály a vezérlőjel által mutatott útvonalon van, a program bírálja felül a vezérlőjelet, kerülje ki az akadályt, majd újra határozza meg a célpontig vezető utat a robot új elhelyezkedését figyelembe véve.

A Lego robot megépítése során csak az alapcsomagban található eszközöket használtam. A kamera szerepét egy átlagos webkamera töltötte be, amely USB porton volt csatlakoztatva egy Linux operációs rendszert futtató számítógéphez. Szükség volt egy Windowst futtató PC –re is, mert a választott programkönyvtár, illetve fejlesztői környezet csak Windows Xp rendszeren működik.

A dolgozatom végére a Lego robotról, az NXT programozási lehetőségeiről, valamint a Bluetooth kommunikációs csatornáról részletes információkkal gazdagodtam, amelyek alapján nagyon sokoldalúnak tartom ezt az eszközt. A sokoldalúsága abban rejlik, hogy szinte minden főbb nyelven lehet rá alkalmazásokat írni, és a magas szintű programozási nyelvek révén a bonyolultabb algoritmusok és az optimalizálások is megvalósíthatóak. A részegységek (motorok és szenzorok) pedig elég precízek ahhoz, hogy egy bonyolultabb fizikai modellezést igénylő projektet is megvalósítsunk (pl. egy kétkereken egyensúlyozó robot).

# Köszönetnyilvánítás

Köszönetet mondok szüleimnek, akik békés családi hátteret és anyagi támogatást nyújtottak tanulmányaimhoz.

Köszönetet mondok témavezetőmnek, dr. Szabó Istvánnak és Juhász Tibornak, akik támogatása, ösztönzése és tanácsai nélkül ez a dolgozat nem valósult volna meg.

Köszönetet mondok a Szilárdtest Fizika Tanszéknek, amiért az eszközöket rendelkezésemre bocsátotta, valamint nyugodt munkát tett lehetővé oldott hangulatú légkörben.

# Felhasznált irodalom


1. Magyar Attila – Robotika
   http://www.aut.vein.hu/oktatok/MagyarA/Rob1.pdf
2. Analysis of the NXT Bluetooth - Communication Protocol :
   http://www.tau.ac.il/~stoledo/lego/btperformance.pdf
3. A LEGO MINDSTORMS robot, NXT technológia:
   http://mindstorms.lego.com
4. Mezzanine User Manual:
   http://playerstage.sourceforge.net/doc/mezzanine-manual-0.00.pdf
5. A Bluetooth technológia:
   http://www.georgikon.hu/mobilkom/bluethoot.htm
6. Wikipedia
   http://hu.wikipedia.org/wiki/Bluetooth
7. Telekommunikáció
   http://vip.tilb.sze.hu/~wersenyi/TKJ.pdf