

TDK dolgozat

Készítette: Veréb László

Nyíregyházi Főiskola

Természettudományi és Informatikai Kar

Nyíregyháza, Sóstói u.31/B.

Foglaltsági háló alapján feltérképező robot

Témavezető: Simon Béláné dr.

Főiskolai tanár

Készítette:

Veréb László

Programtervező informatika

nappali tagozat

I. évf. hallgató

Nyíregyháza 2012

Tartalom

<i>Bevezetés</i>	4
<i>A Mindstorm NXT ismertetése</i>	5
<i>A különböző fejlesztési környezetek ismertetése</i>	8
LEGO Mindstorms NXT-G	8
NXC (Not eXactly C).....	8
LeJOS	9
RobotC	9
Távvezérlés lehetősége.....	9
<i>Térképkészítési folyamatok osztályozása</i>	10
A tárgyak kezelésének módszere alapján.....	10
Metrikus és topológikus térképkészítés.....	10
Szemszög alapján	10
Világ- és robotközpontú térképkészítés	10
<i>Lokalizálás</i>	11
SLAM	11
Kálmán-szűrő	11
Részecske szűrő	11
<i>Kapcsolattartás a külvilággal:</i>	12
<i>Saját kutatás</i>	13
A robot	13
Manőverezés a robottal	13
Két kormányzott és két meghajtó kerék	13
Két különállóan meghajtott és egy támasztott kerék	14
Lánctalppal történő haladás.....	15
Négy lábbal véghezvitt mozgás	15
<i>Szenzorok elhelyezése a roboton</i>	15
A robotom felépítése	16
Egy ilyen típusú robottól elvárt tulajdonságok	16
A saját robotom felépítése.....	16
<i>Egy eredményesen használható teszterület kialakítása</i>	18

Veréb László

Programtervező informatika

I. évf. hallgató

Foglaltsági háló alapján feltérképező robot

Ideális tesztfelület egy nem autonóm térképező robot szempontjából	18
Félig autonóm robotok	18
Első módszer	18
Második módszer	18
Harmadik módszer	19
Teljesen önálló robotok.....	19
<i>Számítási és helymeghatározási módszer</i>	<i>20</i>
<i>Radar létrehozása egy kör feltérképezése</i>	<i>22</i>
<i>Mozgó robot létrehozása</i>	<i>24</i>
Az elmozdulás összehangolása a mérési adatokkal	24
<i>Befejezés.....</i>	<i>25</i>
<i>Felhasznált irodalom</i>	<i>26</i>
<i>Melléklet.....</i>	<i>27</i>
A „radar” forráskódja	27

Bevezetés

Az egyre gyorsuló tudományos és technikai fejlődés végett a közeljövőben a robotok szerepe hatalmasat fog nőni a mindennapjainkban, mint például az internetnek a 90-es évek folyamán. Akkorra kiaknázatlan területek találhatók ezen a területen, hogy hamarosan hatalmas verseny fog kialakulni a piac kézbentartására. Ahogyan fejlődik az ipar és egyre olcsóbbak és olcsóbbak lesznek a robot építéséhez szükséges anyagok úgy fog egyre gyorsabban és gyorsabban beépülni a mindennapjainkba ezek a robotok. Már mostanság is lehet találkozni elvétve egy – egy automata fűnyíró vagy takarító robottal, de ezeknek a használhatósága és ára még nem közelíti meg a ma elvárt színvonalat. Egy nagyon kifinomult szerkezetet nem csak takarítási célokra lehet használni, hanem életmentő célokra is: egy földrengés után az összedőlt épületek romjai között túlélők után kutatva lehet, hogy olyan helyre kell menni ahová más nem tud bemenni csak egy ilyen robot, mivel mélyen a romok alatt van. Ilyenkor nem oldható meg sem a vezetékes sem a rádiós kapcsolat a központtal mivel a törmelékek árnyékolják a rádiókapcsolatot, és az sem lenne túl előnyös, ha esetenként több száz méternyi vezetékkel kellene magával vonszolni egy ilyen robotnak.

Egy ennyire fontos szerepkör betöltéséhez igen komoly tudományos háttérre van szükség mind a robotok felépítése mind az őket irányító szoftver megtervezése folyamán.

Manapság a robotok robbanásszerű terjedésének az egyik legnagyobb akadály a megbízható minden körülmény között biztonságosan alkalmazható multifunkcionális vezérlőszoftver hiánya.

A helymeghatározásnak azért van kiemelt szerepe az autonóm robotok világában mivel az önjáró robot ennek segítségével tudja elhelyezni magát és a számára fontos dolgokat a környezetében. E nélkül az alapvető feltétel nélkül nem képzelhető el semmilyen teljesen automatizált robot sem.

Ebben a munkában elsődleges célom egy olyan NXT alapú robot létrehozása, amely képes egy térképet készíteni közvetlen környezetéről melyet a számítógépen egy összetett térkép létrehozására használhatunk. Ezen robot által szeretném bemutatni a navigáció és térképkészítés alapvető problémáit.

A Mindstorm NXT ismertetése

A LEGO Mindstorm legelső verziója 1998-ból származik, RIS (Robotics Invention System) néven került piacra. Eredetileg oktatási céllal jött létre. A LEGO moduláris építőkockái ideális platformot adtak egy egyszerűen használható, sokcélú készlet létrehozásához és ezzel lehetőséget adtak akár ipari robotok, akár más beágyazott rendszerek modellezéséhez.



Az NXT alapkészletből építhető Alpha Rex nevű robot

Az évek során hatalmas fejlődésen ment keresztül, az NXT 2.0 csomag tartalmaz:

Egy programozható 32 bites téglát, beépített bluetooth-szal és USB csatlakozóval. Programozható mátrixos kijelző, 4 kimeneti, 3 bemeneti, 6-eres digitális csatlakozó. 8KHz-es hangszóró.

6 db AA elemmel vagy akkuval működtethető. A NXT készlet látszólag 4 különböző típusú érzékelőt tartalmaz, valójában ötöt, mivel a motor egyben forgásérzékelő is.

- 3 db szervo motor:

Veréb László

Programtervező informatika

I. évf. hallgató

Foglaltsági háló alapján feltérképező robot

A motorokban beépített forgásérzékelő van, amely 1 fok pontossággal képes mérni az elfordulást.

*A szervómotor*

- Távolság érzékelő:

Az érzékelő cm-ben mér, 0 cm (valójában 5 cm) és 255 cm közötti tartományban nagyjából 2-3 % pontossággal. Sajnos a távolság megállapítása akkor helyes, ha kellő mennyiségű jel verődik vissza a felületről, így igen sok esetben nem érzékel pontosan.

*Az ultrahangos távolságmérő*

- Fényérzékelő és lámpa:

A vörös fényű lámpa kikapcsolható, ekkor csak a környezeti fényt méri. A saját lámpa fényének hullámhossza olyan, hogy többségében pontosabb mérést tesz lehetővé, ezért célszerű bekapcsolva használni.



A fényérzékelő

- Hangerősség érzékelő:
Az érzékelő hangfrekvenciát nem mér, csak erősséget 0 és 100 között. Ne felejtsük el, hogy a mérhető erősség igencsak függ a hangforrás távolságától is. Használata előtt célszerű a háttérzaj szintjét kimérni, és a fölötte megadni a jelszintet.



A mikrofon

- Nyomásérzékelő:
Az állapotában történő változásoknál lehet figyelni a benyomását, az elengedését, illetve hogy klikkelés történt-e. Ez gyakorlatilag benyomás 1, elengedés 0, klikkelés egy 0-1 váltás. Már 33 grammnyi tömeg benyomja, azaz 0,33 N erő kell a jeladáshoz, de erőkar használatával ez csökkenthető. Fontos, hogy a végállásig kell nyomni, csak akkor jelez

Veréb László

Programtervező informatika

I. évf. hallgató

Foglaltsági háló alapján feltérképező robot



A nyomásérzékelő

A különböző fejlesztési környezetek ismertetése

Mivel a LEGO ezt a készletét elsősorban gyerekeknek készíti, ezért a robothoz mellékelt fejlesztési környezet eléggé kezdetleges, elsősorban olyanoknak van kitalálva, akik nem jártasak a programozásban, ezért csak alap dolgokat biztosít a fejlesztéshez. Szerencsénkre a LEGO elérhetővé tette a téglák firmware-jét, mind a hozzá tartozó szoftver specifikációkat, így megnyílt a fejlesztők előtt az út új fejlesztőszoftverek elkészítéséhez.

LEGO Mindstorms NXT-G

Ez az alap fejlesztőkörnyezet, amelyet a csomaghoz kapunk. Grafikus építőelemekből viszonylag könnyen és gyorsan felépíthető vele egy program, de a komolyabb programok nem igazán áttekinthetőek benne, nagy a memóriaigényük és viszonylag lassan futnak.

NXC (Not eXactly C)

Egy ingyenes C változata, az NXC-re épül, mint a neve is mutatja, hasonlít a C nyelvre. Még vannak korlátai, de folyamatosan jönnek ki mind a két nyelvből frissítések és javítások, így valószínűleg hamarosan jól használható lesz. Nem csak az NXT, hanem az elődje, az RCX programozására is alkalmas.

LeJOS

Ez a programozási környezet lecseréli az NXT operációs rendszerét, és egy virtuális gépet hoz rajta létre, amely lehetővé teszi java nyelvű programok futtatását is. Ez a megoldás sem nevezhető teljesen kiforrottnak, de folyamatosan jelennek meg hozzá kiegészítések.

RobotC

Ezt a nyelvet a Carnegie Mellon egyetem robotakadémiája fejleszti. Többek között az NXT készlettel is használható. Ez az egyik legjobban kidolgozott fejlesztőszoftver. Használatához szükséges lecserélni a robotunk gyári firmware-jét. Ez a nyelv is a C nyelvre épül, pontosan úgy, mint az NXC, de vele ellentétben itt ki lehet használni a C nyújtotta funkciók teljes repertoárját. Ezek mellett még képes a lebegőpontos számábrázolásra is. A leghatékonyabb kódot ebben a környezetben lehet fejleszteni, 130-szor gyorsabb kódot lehet ezzel generálni, mint amit az alapsomagban kapott fejlesztőkörnyezettel elérhetünk. Sajnos nem ingyenes, de egy hónapos próbaverziót letölthetjük az internetről.

Távvezérlés lehetősége

Az NXT-t nem csak a saját processzorán futó programok vezérelhetik, hanem USB vagy Bluetooth kapcsolaton keresztül is irányítható. A távvezérlés alatt nem azt kell érteni, hogy egy ember irányítja a robotot, hanem egy telefonon vagy számítógépen futó program is vezérelheti a robotunkat. Erre a feladatra tökéletesen megfelelnek a RobotC nyújtotta lehetőségek, de több más kidolgozott lehetőség is a rendelkezésünkre áll a kapcsolat létrehozásához. Például kiegészítés készült C#-hoz, Java-hoz, Perl-hez, Python-hoz.

Térképkészítési folyamatok osztályozása

A térképkészítés többféle módon minősíthető: vizsgálható, a *tárgyak kezelésének módszere* alapján: metrikus és topológikus a folyamat, vagy azt szem előtt tartva, hogy *kinek a szemszögéből készül*, lehet: világ- vagy robotközpontú.

A tárgyak kezelésének módszere alapján

Metrikus és topológikus térképkészítés

A metrikus vagy más néven háló-alapú térképkészítés a tárgyak számokkal leírható tulajdonságainak segédletével hoz létre egy térképet, ami egy felülnézeti képre hasonlít, lényegében a terület egy kicsinyített mása. Ez a módszer könnyen kezelhető nem érzékeny a zajokra, viszont memóriaigényes, kell a robot pontos pozíciója és nehezen navigálható.

A topológikus térképkészítés pedig azt vizsgálja, hogy el lehet e jutni egyik tárgytól a másikig így egy gráfszerű térkép jön létre. A gráf csúcspontjai a számunkra fontos tárgyak az élek pedig a közvetlen utakat jelképezik. A metrikussal ellentétben ezt nehezebb kezelni és sokkal érzékenyebb a zajokra, de kevés a memória igénye, csak viszonylagos pozícióját kell, hogy tudjuk a robotnak és könnyen navigálható.

Szemszög alapján

Világ- és robotközpontú térképkészítés

A világgözpontú térkép valójában egy nagy koordináta-rendszerben ábrázolja a felfedezni kívánt területet, míg a robotközpontú navigáció a robot szemszögéből. Az első módszerrel nehezebb létrehozni a térképet viszont jobban használható a későbbiek folyamán, amikor be szeretnénk járni a terepet, a második módszerrel a térkép könnyebben létrejön, de jóval nehezebb használni azt.

Lokalizálás

Lokalizálás az, amikor valaminek a helyét térkép szerint keressük meg.

Az autonóm robotok számára nagyon fontos, hogy ismerje a saját helyét és képes legyen elnavigálni egy másik helyre.

Az intelligens lokalizálásnak és navigálásnak legáltalánosabb (legelterjedtebb) módszere, hogy a szenzor leolvasásokat és valamilyen zártláncú mozgás-visszacsatolást kombinálunk a térkép használatával. Egy robotnak sokkal többet kell csinálnia, mint saját magát lokalizálni, neki kell a térképet is elkészítenie. A szimultán lokalizálást és térképkészítést nevezik a robotikában SLAM¹-nek.

SLAM

A SLAM (Simultaneous Localization And Mapping), azaz a Szimultán Helymeghatározás És Térképezés a robotika egyik legfontosabb problémája volt, amire megoldást csak a 90-es évek közepén tudtak találni a kutatók. A probléma lényege az, hogy a robotnak egy ismeretlen kezdőpozícióból kezdve kell egy fokozatosan növekvő térképet létrehozni, amely alapján meg tudja állapítani a saját helyzetét a térképen. Már 1986-ban meghatározta a kutatói közösség a problémát, ekkortól kezdtek valószínűségi módszerekkel kísérletezni a robotikában is. Nem sokkal később meg történtek az első próbálkozások a Kalman-szűrő alkalmazására is. Az áttörés 1995-ben következett be, amikor rájöttek, hogy a kombinált térképezési és helymeghatározási probléma egyetlen becslési feladatként is felfogható. Mostanra a SLAM megoldottnak tekinthető, már csak kisebb finomítás munkák vannak hátra.

Kálmán-szűrő

Ennek segítségével a különféle adatsorokból kiszűrhető a „zaj”, s hiányos információk esetében is – rekurzív paraméterbecsléssel – megállapítható az adott rendszer időállapota.

Részecske szűrő

A Kálmán-szűrőn kívül még használatos módszer a particle filter (részecskeszűrő) is. Több szakterületen is ismert és alkalmazott eljárás. A részecskeszűrő alapja, hogy minden állapotban részecskék segítségével számítunk közelítést. A részecskék követik a rendszer dinamikáját, és állapotról állapotra frissülnek.

¹ Egyes szakirodalmakban Concurrent mapping and localization (CML) néven hivatkoznak rá

Kapcsolattartás a külvilággal:

A kapcsolattartás a külvilággal a robot szenzorok segítségével végzi, ezek lesznek, az érzékszervei miközben végrehajtja a feladatát. Az érzékelők felé érkező ingereket külső hatások befolyásolhatják, torzíthatják, ezeket a hatásokat nevezzük zajnak.

Mint korábban utaltam rá a Mindstorm NXT-nek többféle érzékelője van. Ezek közül legpontosabb a compass nevű szenzor, amely iránytűként funkcionál, ennek segítségével meg lehet határozni a robot elfordulását a kezdeti állapotához képest. Célszerű minél távolabb helyezni a motoroktól, mivel a mágneses térük befolyásolhatja a működését. A motorok tachométere méri a kerekek elfordulását, ezáltal információt kapunk a megtett útról, de ez már nem annyira pontos, mint a fentebb említett „tájéoló”. A szonár szenzor a robotunk „szeme”, meghatározhatjuk vele a távolságunkat egy adott ponttól. Sajnos ez a szenzor a legpontatlanabb a többihez viszonyítva, mert a környezet zajai igen hátrányos módon befolyásolják a működését.

Választásom azért esett a foglaltsági hálón alapuló feltérképezési módszerre, mivel egy viszonylag könnyen írható forrásállományú, a vezérlőrendszert lehet létrehozni a segítségével, ami nem annyira érzékeny a zajokra, mint a többi módszer

Saját kutatás

Ebben a fejezetben elsősorban azt szeretném leírni, hogy milyen eszközök és tudás kell egy ilyen projekt létrehozásához. Sajnos a projektet teljes egészében nem tudtam létrehozni, de részletekben megcsináltam a nagy részét legfőképpen idő, és anyagihiány miatt, de így is rengeteg információra tettem szert a robot működéséről amelyet a későbbiekben szeretnék megvalósítani.

A robot

Korábban szerzett tapasztalataim alapján tudom, hogy milyen kritériumoknak kell megfelelnie egy ilyen robotnak. Fontos számára a pontosan manőverező képesség, stabil mozgás, szenzorok pontos beállítása. A LEGO által gyártott hardware -nak megvannak a saját érdekes hibái, amelyek egy részére megoldást nyújtana a SLAM, de sajnos nem mindre, mivel csak a helymeghatározásban van segítségünkre ez az eljárás. A tachométere a szervomotoroknak sajnos nem ad vissza mindig pontos értékeket, amely több ízben is gyengítené a konstrukciónk manőverezési és feltérképezési képességét.

Manőverezés a robottal

Több lehetőség is előttünk áll a robot mozgatására ezek közül szeretnék felsorolni egy pár általam ismert és megépített módot.

Két kormányzott és két meghajtó kerék

Ez a megoldás az autókban jól ismerik, és hátsó kerékmeghajtásnak nevezik. A két első kereket egy motor segítségével tudjuk elforgatni, amely a kormányzást biztosítja, de ezek nem rendelkeznek meghajtással. A gyorsításról a két hátsó kerekünk gondoskodik, amelyeknek egy közös tengely által meghajtott motorral tudunk energiát adni. Ez a konstrukció nem előnyös a feltérképezés szempontjából mivel ennek nagyobb a fordulóköre az általam favorizált megoldáshoz képest, és ugyanannyi motort használ. Tapasztalataim szerint ezeken a hibákon felül még az első kerékkormányzás sem megfelelően pontos mivel nincs meg a megfelelő tapadása a kerekeknek és ezért nem lehet pontosan kormányozni.

Veréb László

Programtervező informatika

I. évf. hallgató

Foglaltsági háló alapján feltérképező robot



A két hajtott és két kormányzott kerekű robot

Két különállóan meghajtott és egy támasztott kerék

Az alap tervem a robot építésére az volt, hogy két darab meghajtó kerék lenne elöl és egy támasztókerék tartaná hátul a robot alvázát. Ez a konstrukció több szempontból is előnyös egy ilyenfajta robot számára mivel a két meghajtott kerék elég nagy mozgásszabadságot biztosít a mozgásban. A szerkezetünk kormányzását a kerekek eltérő fordulatszámával tudnánk megoldani, ha az egyikoldalon megnöveljük a fordulatszámot, akkor ívesen lehet fordulni, viszont ha csak az egyik kerekünket, hajtjuk meg akkor akár saját maga körül is meg tud fordulni. A pontos fordulás érdekében mivel a tachométerek pontossága nem a legjobb használhatunk egy compass szenzort, amely iránytűként funkcionálna, így biztosan a kívánt szögben tudnánk elfordulni, viszont ebben az esetben gondoskodnunk kell a szenzor megfelelő elhelyezéséről mivel a motorok zavarhatják a compass szenzorunkat.



A két meghajtott keréssel rendelkező robot

Veréb László

Programtervező informatika

I. évf. hallgató

Foglaltsági háló alapján feltérképező robot

Lánctalppal történő haladás

Ez a technika nagyon hasonlít az előbb felvázolt módszerhez, annyi különbséggel, hogy nincs támasztókerék csak a robot két oldalán végigfutó talp. Az irányítása is nagyon hasonlít a fenn említett eljáráshoz. Ezt a módszert nem egyenes talajon érdemes használni mivel jóval nagyobb stabilitást biztosít.

Négy lábbal véghezvitt mozgás

Ezt a meghajtási módszert csak érdekességként említem meg mivel ezt a módszert magam fejleszttem és még nem volt időm tökéletesíteni. Lényegében két darab motorral végeztem a lábak meghajtását. A robot két oldalán elhelyezett lábakat egy motorral hajtottam meg amelyekhez áttételezéssel juttattam el az energiát. Ezt a konstrukciót sajnos nem lehet ilyen feladatra használni mivel a mozgását egyelőre nem lehet megfelelően szabályozni, és a fogaskerekek illeszkedése sem volt a legtökéletesebb. Bár ha kiegészítettem négy fix támasztékkal az jelentősen javított a stabilitásán, de nem eleget ahhoz, hogy ilyen feladatra alkalmas legyen.



A lábbakkal mozgó robot

Szenzorok elhelyezése a roboton

Mielőtt belekezdünk egy robot építésébe, fontos eldöntenünk, hogy milyen érzékelőket teszünk egy robotra. Több szempontot is figyelembe kell, vegyünk. Szem előtt kell tartanunk a robotunk képességeit, hiába szeretnénk öt érzékelőt használni, ha csak négyet tud kezelni robotunk. Esetleg az érzékelők zavarhatják is egymást a mérésben, így helytelen értékkel becsapva minket.

Gépünk elsődleges célja egy térkép elkészítése egy ismeretlen területről. Ehhez pontosan kell, ismerje a saját és a környezetében található tárgyak helyét. Ezért nagyon precíz szenzoradatok kellenek, amiben ki vannak küszöbölve a környezet zajai, a szenzorok

Veréb László

Programtervező informatika

I. évf. hallgató

Foglaltsági háló alapján feltérképező robot

mérési hibái. Ezekre a problémákra kellően figyelmes tervezéssel megoldásokat lehet találni. Elsődleges probléma egy NXT típusú robottal, hogy csak négy szenzort lehet egyszerre csatlakoztatni hozzá. Az eddig tanulmányozott algoritmusok pedig több érzékelőt követelnek meg tőlünk. Erre a problémára szerintem a leoptimálisabb megoldás az, ha egy érzékelőt több célra is felhasználunk, vagy ha számos ugyanolyanra lenne szükség, akkor egyet használjuk fel többször. Ezeknek az algoritmusoknak fontos, hogy a kiindulási pozíciókhoz képest jól el tudjuk magunkat helyezni a térben. A pontos eredmények érdekében mindenképpen fontos tudnunk, hogy az eredeti pozíciókhoz képest milyen irányban állunk. Ebben megfelelő segítséget tud nyújtani számunkra a compass szenzor, amely pontos adatot ad a robotunk aktuális irányáról, de ez nem elég, hogy meg tudjuk állapítani a korrekt helyzetét, muszáj tudnunk az általa megtett út hosszát is. Ha el tudjuk magunkat pontosan helyezni a térképen, és ismerjük a körülöttünk lévő tárgyak távolságát, és irányunkhoz viszonyított szögét akkor ki tudjuk számolni a térképen is a helyét. Ahhoz hogy ezt a szöget ismerjük, vagy sok stabil érzékelőt, vagy egy mobilt kell alkalmazunk. Én az utóbbi megoldásra szavazok, mivel az általam választott modellnek csak korlátozott számú port található.

A robotom felépítése

Egy ilyen típusú robottól elvárt tulajdonságok

Egy ilyen robotnak sok elvárásnak kell megfelelnie. Fő szempontok azonban a jó manőverező képesség és stabilitás.

A saját robotom felépítése

Az NXT által biztosított lehetőségek meglehetősen nagy gátat szabnak a tervezés közben. A fő modul csak három darab motornak biztosít helyet, így egy több kettőnél több külön meghajtott kerékkel rendelkező robot felépítése szinte lehetetlen, mivel gondoskodnunk kell a távolságmérő szenzorunk forgatásáról is. A szonár forgatása azért szükséges, mert egy érzékelőt használok az akadályok felmérésére. Többet hiába is akartam használni, nem használhattam, mivel zavarják egymást az ultrahanggal működő szenzorok. A távolságmérőt érdemes a robot legmagasabb pontján elhelyezni, azért hogy egy ne takarjon a látószögéből az érzékelőnek, és egy teljes kört fel lehessen fedezni. Két meghajtott kerék és hátul egy csúszó talp segítségével biztosítjuk a megfelelő mozgást. A csúszó talpat azért részesítettem előnyben a támasztókerékkel szemben, mert így meg tud fordulni saját tengelye körül a szerkezet. Közvetlenül a motorok fölött helyezkedik el az NXT téglá, amelyre ráépítettem a szervómotort, amely a szonárt mozgatja. Ezzel a felépítéssel biztosítottam minden fontos tulajdonságot, melyekre szüksége van egy ilyen robotnak,

Veréb László

Programtervező informatika

I. évf. hallgató

Foglaltsági háló alapján feltérképező robot

ezen felül sikerült alacsonyan tartanom a súlypontját, ami azért fontos, mert így sokkal nehezebben borul fel a robot.



Az előbb bemutatott robot

Egy eredményesen használható tesztterület kialakítása

Egy nem teljesen önálló robotnak, amely csak a feltérképezéssel törődik más tesztpad ideális, mint egy teljesen független robotnak. Mivel különböző feladatoknak kell megfelelnie, és ezeket a követelményeknek más módon tudunk megfelelni. Viszont az összes esetben nagyon fontos a pályának néhány tulajdonsága, amelyet nem befolyásol a robotunk rendeltetése. Fontos, hogy vízszintes legyen a talaj, nehogy rossz értéket kapjunk. Az akadályoknak is fontos az alakja és anyaga, mivel egy gömbölyű tárgyról másképp érkeznek vissza a jelek, mint egy egyenesről.

Ideális tesztfelület egy nem autonóm térképező robot szempontjából

Az ilyenfajta robotok csak a térképkészítéssel törődnek, a helyzet meghatározást nekünk kell biztosítani a számukra. Ilyenkor manuálisan megadhatjuk a pozícióját a robotunknak. Ebben az esetben nem kell törődnünk a robotunk helyváltoztatásának pontatlanságaival, mivel ilyenkor mi a felhasználók mondjuk meg neki, hogy hol van. Egy ilyen tesztpadnak a számunkra könnyen beláthatónak kell lenni, hogy meg tudjuk mondani a robotunknak, azt hogy éppen hol tartózkodik, esetleg parancsokat tudjunk neki adni, hogy merre menjen.

Félig autonóm robotok

Ezek a robotok kis „stiklivel” működnek, mivel itt már részben saját maguk, vagy teljesen egyedül határozzák meg a pozíciójukat, de egy kis rafináltsággal fűszerezve. Több példával is szeretnék élni ebben a témában, mert nagyon sok lehetőség van ilyen megoldásokra.

Első módszer

Egy külső pozícióból kamerával figyeljük a robot mozgását, melyet pontos adatokká tudunk alakítani bizonyos képfeldolgozó módszerek segítségével. Össze tudjuk hasonlítani a robot jelenlegi helyzetét a robot előző helyzetével és ennek alapján tudjuk meghatározni jelenlegi pozícióját. Ezzel a módszerrel egy pontos helymeghatározási módszert lehet létrehozni, viszont így a robot függ külső hardware elemektől is.

Második módszer

Egy vezetővonal festése a padlóra. A robotnak van egy vonal felfestve, amin megszakítások találhatóak, amik a mérési pontokat jelképezik. Ha megfelelően alakítjuk ki

Veréb László

Programtervező informatika

I. évf. hallgató

Foglaltsági háló alapján feltérképező robot

a mérési pontok helyét, akkor nagyon könnyen tudathatjuk a szerkezettel, hogy milyen pozícióban van, viszont ha át akarjuk rendezni tesztpadunkat, akkor a vezetővonalat is át kell alakítanunk. Ebben az esetben a robotunk már nem függ külső hardware eszközöktől, de ilyenkor kell egy „térképet szerkesztenünk” a térképkészítéshez.

Harmadik módszer

Vezetővonalak festése a padlóra. Készíthetünk egy négyzetrácsot a padlóra és ennek a segítségével határozhatjuk meg a robotunk helyét, 1 vonal egy lépés a mátrixban, ebben az esetben egy iránytűvel tudnánk meghatározni, hogy a robotunk milyen irányba halad, tehát milyen irányban kell haladnunk a mátrixunkban. Az előző módszerhez képest itt nem befolyásolja a négyzetrácsunk kinézetét az akadályok elhelyezkedése, egész egyszerűen, ha akadályt észlel, a robot kikerüli.

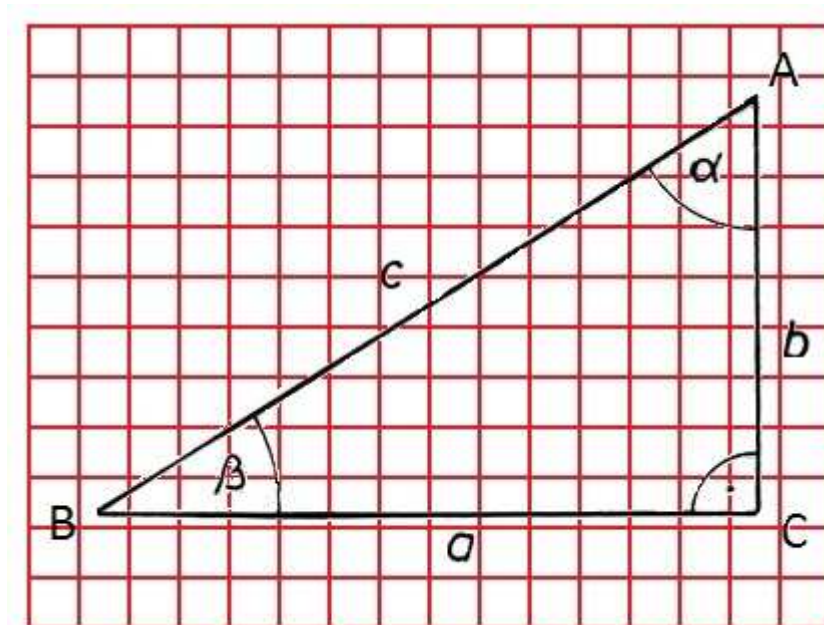
Teljesen önálló robotok

Mivel ebben az esetben a robot a kezdeti pozícióját teljesen egyedül frissíti, ilyenkor semmilyen külső beavatkozásra nincs szükség. Ezért a terület kialakítása a padló szempontjából annyira nem fontos, viszont előnyös, ha vezetővonalakat teszünk a padlóra, mivel az akadályok helyének a meghatározása és elhelyezése sokkal könnyebb számunkra.

Számítási és helymeghatározási módszer

Ismert kezdőpozícióval kell rendelkezünk ahhoz, hogy sikeresen el tudjuk készíteni ezt a kísérletet. A helyzetünk megállapítása az első lépés után a lehető legkézenfekvőbb módon történik, a robotunk szervomotorának tacho métere segítségével. Ha tudjuk, a kerekünk sugarát abból megtudhatjuk, hogy mekkora a hossza a kör kerületének. Ha ezt az értéket elosztjuk 360-nal, akkor megtudjuk, hogy egy fok mekkora távolságot jelent a próbaasztalunkon. Miután ismerjük, a helyzetünket el tudjuk helyezni az érzékelt akadályokat két érték segítségével, amelyeket a robot szolgáltat nekünk.

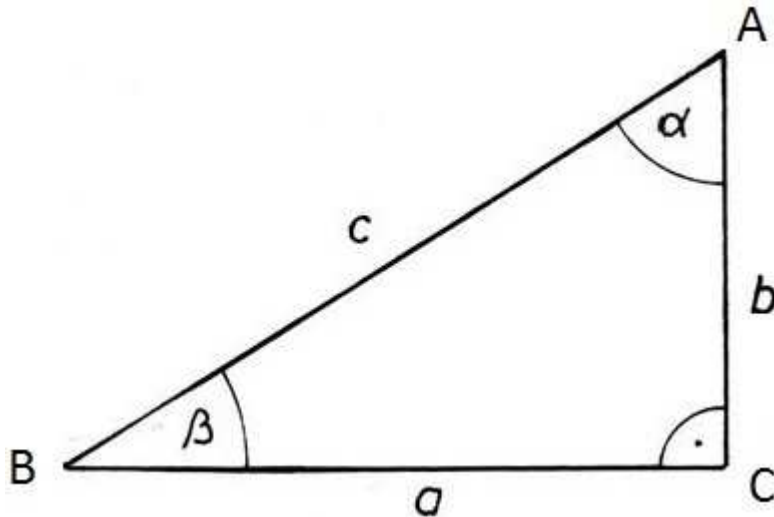
Az első érték a robotunk és az akadály távolsága, a második pedig, az a szög, amelyet bezár az akadályunk a jelenlegi irányunkhoz képest. Ennek alapján egyszerűen szinusz segítségével ki tudjuk számolni a pozíciót.

**1. ábra**

Tételezzük fel azt, hogy a B pontban helyezkedik el a robotunk. Az A pontban található az akadály. Az a oldal a robotunk haladási iránya. A c oldal az az egyenes, amely a robotunk és az észlelt tárgy között helyezkedik el. A b oldal pedig az oldalirányú eltérést reprezentálja a robothoz képest. Ha ezt megfigyeljük, akkor látjuk, hogy ez egy derékszögű háromszöget ábrázol. A robot érzékeli a BA szakasz hosszát, és a β szög nagyságát:

A c oldalt a szonár szenzorjának segítségével érzékeli, amely egy centiméterben mért adatot ad vissza. A β szöget pedig a szenzor elfordulási szöge adja, amelyet egy szervomotor tachométere fokokban állapít meg. Ezeknek az adatoknak, és a derékszögű háromszög

tulajdonságainak segítségével könnyedén ki lehet számolni az akadály elhelyezkedéséhez szükséges adatokat.



2. ábra

A derékszögű háromszögben a hegyesszöggel szemközti befogó és az átfogó arányát a szög sinusának nevezzük. $\sin\alpha = \frac{a}{c}$, $\sin\beta = \frac{b}{c}$

Ezeket a képleteket elemi átalakításokkal nekünk megfelelő alakra tudjuk hozni: $a = \sin\alpha \cdot c$
 $b = \sin\beta \cdot c$

Az α szöget pedig a következő módon kapjuk meg: $\alpha = 90 - \beta$

Ha a kapott értékeket elosztjuk egy rácunk oldalának hosszával, akkor megkapjuk, hogy melyik hely a foglalt.

Radár létrehozása egy kör feltérképezése

A robotunk szempontjából nagyon fontos, hogy kidolgozzunk egy megbízható algoritmust, amely jól ábrázolja a körülötte lévő tárgyak helyzetét. Ha felfedezzük a közvetlen környezetünket és tudjuk mérni a robotunk elmozdulását, akkor a környezeti képekből összetudunk rakni egy komplex képet amit már térképnek nevezhetünk.

Jelen esetben a robot nem mozog mivel, csak egy szűk területet szeretnénk megfigyelni. Az előbb leírt eljárásnak a segítségével határozzuk meg az akadályok viszonyított helyzetét.

A robot forráskódjának elkészítéséhez a RobotC nevű programot használtam. A Carnegie Mellon egyetem fejlesztette fizetős környezetet azért részesítettem előnyben a többi ingyenessel szemben mivel, a többivel összehasonlíthatatlan támogatást biztosít, az általa generált állományok nagyságrendekkel gyorsabban futnak a többi C nyelven programozható megoldáshoz képest, és az iskola birtokol licenszet a használathoz.

A forráskód szerkesztése során felmerült pár váratlan probléma az előző fejezet felvázolt algoritmussal kapcsolatban.

Az első az volt, hogy az algoritmus nem adott helyes értéket vissza, ha a β szög nagyobb volt, mint 90 fok. Erre a megoldást egy előltesztelési ciklussal találtam, amely addig csökkentette a β szög értékét, míg 90 fok alá nem ért (1. forráskód).

```
while(szb>90)
{
    szb=szb-90;
}
```

(1. forráskód) szb a β szög értéke fokokban mérve

A következő probléma az észlelt tárgyak megjelenítésében jelentkezett. A robot kijelzőjén reprezentáltam az akadályok helyzetét. Mivel a β szög értékét le kellett csökkentenem ahhoz, hogy ki tudjam számítani a pontok helyét 90 fok alá így minden bolyát a kör első negyedébe helyezett az algoritmus. A megoldást erre sok felesleges számítási munka nélkül szerettem volna megoldani így, több egymásba ágyazott if függvényt alkalmaztam. Mivel minden új negyedbe lépéskor az eredeti szög nagyobb mint az előző végén így

fokozatosan növekvő feltételekkel ezt viszonylag kis számolási igény növekedéssel megtudtam oldani. (2. forráskód)

```
if(sze>90)
{
  a=a*-1;
  if(sze>180)
  {
    b=b*-1;}
  if(sze>270)
  {
    c=b;
    b=a*1;
    a=-1*c;}
  if((sze>90) && (sze<180))
  {
    c=b;
    b=a*-1;
    a=c*-1;}}
```

(2. forráskód) sze az eredeti szög a az y tengely irányába mutató vektor
b az x tengely irányába mutató vektor
c a cseréhez szükséges vektor

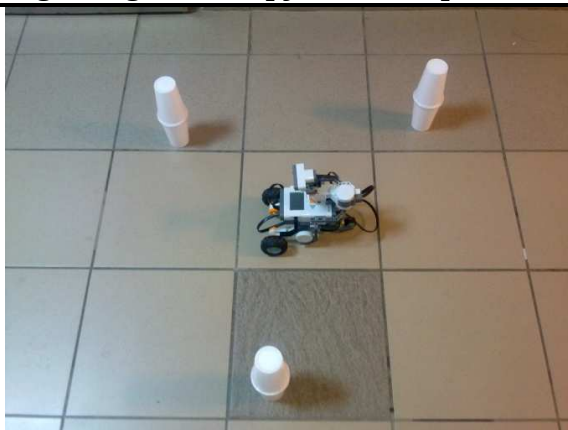
Ha tudjuk a helyzetét az érzékelt tárgyaknak, akkor azokat valahogyan meg is kell jeleníteni. Erre a célra tökéletesen megfelel a brick, amely egy 100*60-as felbontású fekete – fehér LCD kijelzővel van szerelve. Létrehoztam egy kört a képernyő közepén, amely a szenzor érzékelési határát jelöli. Lektorlátoztam az érzékelő hatótávolságát 60 centiméterre, mivel ezen a távolságon túl már nem adott vissza megfelelően pontos értéket. Az algoritmus által kapott értékeket hozzá kell igazítani a kör középpontjához, mivel alaphoz a kijelző nullpontjára adja az értékeket. Úgy határoztam meg a kör pozícióját, hogy teljesen a kijelző közepén helyezkedjen el, és teljes magasságában elfoglalja azt. Ez azt jelenti, hogy egy pixel egy 2x2-es négyzetnek felel meg a valóságban, ami megfelel egy egyszerűsített foglaltsági hálónak, amely két értéket jelenít meg az akadályok reprezentálására.

Veréb László

Programtervező informatika

I. évf. hallgató

Foglaltsági háló alapján feltérképező robot



A robot munka közben



A kapott kép

[A teljes forráskód a Mellékletben van feltüntetve.]

Mozgó robot létrehozása

Az előző fejezetben bemutatam egy olyan robotot, amely képes felfedezni a környezetét. Ettől már csak egy lépésre van egy mozgó robot létrehozása, amely már nem csak egy helyben áll. A mozgásról a meghajtó kerekek gondoskodnak. Valamilyen szinten gondoskodnak a mozgás méréséről is. Ha ismerjük az elmozdulást és el tudjuk menteni a mérési adatokat is akkor már csak össze kell hangoljuk a mérések képeit az elmozdulással, majd megjeleníteni azokat egy grafikonon. ezzel elkészíthetünk egy térképet amely a felfedezett területet átalakít egy kétdimenziós koordinátarendszerben ábrázolható képpé.

Az elmozdulás összehangolása a mérési adatokkal

Ha feltételezzük, hogy pontos adatokat kapunk a robot mozgásáról, akkor könnyedén ki tudjuk számolni a tárgyak egymáshoz viszonyított helyzetét. Ezt a következő módon tehetjük meg: Mivel az alap algoritmus mindent az origó ponthoz viszonyít így csak az origó értékét kell megfelelően növelnünk vagy csökkentenünk, hogy meg tudjuk jeleníteni az adatokat például egy táblázatkezelőben.

Erre a feladatra egy egyszerűbb programot fejlesztettem, amelyben egy dimenzióban mozdul el a robot, és a távolságérzékelővel mért adatokat, rögzítem egy fájlba a robotC által biztosított AddToDatalog paranccsal.

Befejezés

A kutatásom során rengeteg új információt szereztem a térképkészítéssel és lokalizációval kapcsolatban. Ezen felül részletes betekintést kaptam az NXT felépítésébe és tulajdonságaiba. A robot felépítése alkalmas az eredményes térképkészítéshez, de sajnos az érzékelők mérési pontatlanságai akkora gátat szabnak mozgásban és érzékelésben, hogy egy teljesen autonóm robot létrehozása szinte teljesen lehetetlen. Az NXT egy nagyon jól megtervezett alap mindenféle célra megfelelően használható, de nem szabad elfelejtenünk, hogy eredetileg egy gyerekjátéknak szánták.

Sikeresen kidolgoztam egy algoritmust, amely megadja a viszonyított helyét az észlelt akadályoknak, és megjeleníti a robot kijelzőjén. Kialakítottam egy stabil bluetooth kapcsolatot, amelyen keresztül vezeték nélkül is lehet mérési adatokat fogadni és parancsokat küldeni. Ezenfelül létrehoztam egy olyan szoftvert amely egy dimenzióban autonóm módon készít térképet. Mindent egybe véve részben sikeresnek mondható a munkám mivel egy egyszerűsített foglaltsági hálón alapuló lokális térképet létre tudtam hozni, és egy összetett térképet is amely egy dimenziós.

Felhasznált irodalom

Dissanayaka, M., Newman, P., Clark, S., Durrant-Whyte, H., and Csorba, M. : A solution to the simultaneous localisation and map building problem.

IEEE Robotics and Automation 17, 3 (June 2001), 229-241.

Hugh Durrant-Whyte , Tim Bailey :

Simultaneous Localization and Mapping: Part I

IEEE Robotics and Automation (June 2006), 99-108.

Hugh Durrant-Whyte , Tim Bailey :

Simultaneous Localization and Mapping: Part II

IEEE Robotics and Automation (September 2006), 108-117.

<http://www.muszeroldal.hu/MMK/nr68/vogel.html>

<http://www.irobot.com/>

<http://129.96.12.107/confpapers/CRPITV52McNally.pdf>

<http://www.jataka.hu/rics/makog2004/makog2004.pdf>

http://www.nhit-it3.hu/index.php?option=com_content&view=article&id=6481%3Ait3-2-2-4-u&catid=63%3Aautonom-mobil-robotok&Itemid=913&lang=hu

http://www.google.hu/url?sa=t&rct=j&q=&esrc=s&source=web&cd=9&ved=0CIQBEBYwCA&url=http%3A%2F%2Fwww.fzolee.hu%2Fframework%2Ffiles%2F2006_V_Helymeghatarozas.ppt&ei=4XOJT8aAGYHStAaz0qTqCw&usg=AFQjCNGUWwrevUoEQvww8sRY1-nG2K9bTg&sig2= AzR7cpge35hMHse0kd3Xg

<http://forums.trossenrobotics.com/tutorials/introduction-129/an-introduction-to-mapping-and-localization-3274/>

Melléklet

A „radar” forráskódja

```
#pragma config(Sensor, S1, sonar, sensorSONAR)
#pragma config(Motor, motorA, jobb, tmotorNormal, PIDControl, encoder)
#pragma config(Motor, motorB, ball, tmotorNormal, PIDControl, encoder)
#pragma config(Motor, motorC, radar, tmotorNormal, PIDControl, encoder)

task main()
{
float x,y;

int i, a=0, b=0, c, sza, szb, sze;//a = a oldal hossz, b = b oldal hossz, c = c oldal hossz, szb =
beta szog, sza = alfa szog, sze =eredeti szog

eraseDisplay();
nxtDrawEllipse(20, 60, 80, 0);
nxtSetPixel(50, 30);
nMotorEncoder[radar] = 0;
while(nMotorEncoder[radar] <= 360)
{
motor[radar] = 5;

c=(SensorValue(S1)/2);
if(SensorValue(S1)<60)
{szb=nMotorEncoder[radar];
sze=szb;
while(szb>89)
{
szb=szb-89;
}
```

```
sza=90-szb;

y=sinDegrees(sza)*c;
x=sinDegrees(szb)*c;

a=y/1;
  =x/1;

if(sze>90)
{
a=a*-1;
if(sze>180)
{
b=b*-1;
}
if(sze>270)
{
c=b;
b=a*1;
a=-1*c;
}
if((sze>90) && (sze<180))
{c=b;
b=a*-1;
a=c*-1;
}}
nxtSetPixel(50+b, 30+a);
}}

motor[radar] = 0;
while(nMotorEncoder[radar] >= 0)
{motor[radar] = -20;}
motor[radar] = 0;
wait10Msec(3);
wait10Msec(3000);}
```

Az autonóm robot forrása

```
#pragma config(Sensor, S1, sonar, sensorSONAR)
#pragma config(Motor, motorA, jobb, tmotorNormal, PIDControl, encoder)
#pragma config(Motor, motorB, ball, tmotorNormal, PIDControl, encoder)
#pragma config(Motor, motorC, radar, tmotorNormal, PIDControl, encoder)
//*!!Code automatically generated by 'ROBOTC' configuration wizard !!*//
```

```
task main()
{
    int tav, i;
    for(i=0;i<4;i++){
        motor[ball]=50;
        motor[jobb]=50;
        wait10Msec(100);
        motor[ball]=0;
        motor[jobb]=0;
        tav = SensorValue(S1);
        AddToDatalog(tav);
        eraseDisplay();
        nxtDisplayTextLine(7,"%d",tav);
    }
    SaveNxtDatalog();
}
```