Statikus bluetooth kommunikációs láncon alapuló, multi-robot területfelfedező algoritmus

Pásztor Attila, Kovács Tamás
Kecskeméti Főiskola GAMF Kar, Informatika Tanszék

Összefoglalás: A cikkben bemutatunk egy általunk kifejlesztett területfelderítő algoritmust, mely robosztus, sok robotból álló rendszerek esetén is működik. A robotok bluetoothon keresztül kommunikálnak, kommunikáció szempontjából lánccá szerveződnek, a lánc egyik vége rögzített és a felderítés alatt kényszer feltétel az, hogy a szomszédos robotok semmiképp sem távolodhatnak egymás kommunikációs hatósugarából. Az írásban egyszerűen bizonyítjuk, hogy az általunk készített algoritmus optimális akadálymentes területeken, azaz a lehető legnagyobb területet fedezi fel a lehetséges legkisebb időráfordítással. Bemutatunk egy tesztet, melyben összehasonlítjuk az általunk ajánlott eljárást két referencia eljárással, falszerű akadályokkal ellátott területeken is. A cikkben az általunk készített algoritmust nem csak képernyő szimulátorral teszteltük, hanem implementáltuk valós Dual NXT robotokból álló robotcsoportra is.

Abstract: In our paper an area exploration method is presented based on the static linear communication network above. The robots are communicated via bluetooth and are organized a communication chain. One of the end of robot chain is fixed during the exploration and the other constraint is that not allowed for the neighbor robots moving away from the range of other. With a simple proof, we have shown that the proposed (fixed chain-like team) exploration method is optimal in the obstacle-free case under the constraint of the connectivity with the base station . It was found that the proposed method performs better than the chosen reference methods in the case of zero or low obstacle density . The method was tested by computer simulations and real Dual NXTs for various obstacle configurations and densities..

Kulcsszavak: robotraj, területfelderítés

Keywords: robot swarm, area discovering

1. Bevezetés

Napjainkban a raj intelligencia és a mobil robotok kooperációja területeken jelentős fejlődésnek lehetünk szemtanúi. Az élet számos területén jelentkező problémát lehet hatékonyan megoldani egy csapat autonóm mobil robot segítségével. Tipikus és intenzíven vizsgált probléma egy ismeretlen terület felderítése, a terület feltérképezése, vagy különböző célobjektumok megkeresése. Egy ilyen jellegű feladatnak az optimális megoldási stratégiáit többnyire az előírt környezeti feltételek és a robotok képességei határozzák meg.

A problémakör ismertetése

Szinte az összes, robotcsoportokkal megvalósított terület felderítési algoritmus a „határ” elven alapul, azaz, azon hogy milyen a határvonal a felderített és felderítetlen területek között. Az alapötlet az, hogy a robotokat úgy kellene irányítani vagy a felderített és felderítetlen területek határán tartani, hogy a robotcsoport elmozdulási költsége minimális, míg a felderített területnagysága maximális legyen. Az alábbi algoritmusok, ennek elérése érdekében egy hasznossági függvényt használnak, amely növeli az információs nyereséget és csökkenti a költséget, ill. a felderítési időt.

Simons és társai [1] egy „központosított” eljárást alkalmaztak, ahol egy bázisállomás kiszámítja az optimális csoport–mozgást és vezérli a robotokat. A robotok egymástól függetlenek, különállóak, a hasznossági függvény ennek megfelelően van az algoritmusba beépítve.

Burgard és társai [2] egy egyszerű központosított algoritmust javasoltak. Ha a robotcsoport eredetileg folytonos kommunikációs klasztere több kisebb részre szakad szét úgy, hogy a különböző klaszterek nem tudnak kommunikálni egymással, akkor minden különálló klaszter egyedileg folytatja a felderítő algoritmust. Amikor minden robot elszigetelődik egymástól és a felderítő algoritmust magukban végzik, az eljárás átcsap egy decentralizált rendszerbe.

Sheng és társai írásában [3] egy szeparált kommunikációs klaszter problémájának megoldását mutatja be. A robotok egyedi felfedező mozgást hajtanak végre, de kénytelenek rendszeresen kommunikációs kapcsolatot felvenni a társaikkal, így a feladat kiosztási eljárás figyelembe veheti az egész csapatot. Ennek következtében, a felderítési idő jelentős részében a csoport tagjai kívül esnek a többi robot kommunikációs tartományán.

Vazquez és Malcolm [4] egy decentralizált eljárást javasolt, ahol a robotok egyénileg végzik az algoritmusaikat, felelve a terület felfedezéséért a kommunikáció fenntartása és az ütközésfigyelés mellett.

Y.Pey és társai [5] írásukban bemutattak egy eljárást, ahol a csoport mozgás költségét egy teljes lépéssor (a csoport összes tagjának mozgása) ideje adja, két egymást követő csoport pozíció között. Az algoritmusuk központosított, és kapcsolatot biztosít a kezdő állomással minden egyes csoport-mozgás végéig, habár a migrációs fázis közben a kapcsolat nem garantált.

Dahl és társai [6] által publikált cikkben a feladatok felosztását egy ütemezési problémaként kezelik, ezt alkalmazzák bonyolult ipari gyártási folyamatokban is. A megoldásuk heterogén robotcsapatok esetén, ahol a robotok különböző képességűek, figyelembe vesz különböző kijelölt feladatokat (pl. felderítés).

Mivel a robotok fizikai kiterjedéssel rendelkeznek, nem pedig pontszerűek, a legtöbb idézett eljárás figyelembe veszi az ütközés-elkerülés problémáját is a robotok között. Az elméleti algoritmus szintjén ez egyszerűen kivitelezhető egy kifejezés hozzáadásával a költség függvényhez, amely bünteti azokat a konfigurációkat, amelyekben a robotok túl közel mennek egymáshoz. A gyakorlatban azonban a robotokra is szükséges implementálni egy helyi, függetlenül dolgozó tulajdonságot, ami jelzi, ha az ütközés veszélye fennáll. Az általunk készített algoritmus tesztelésénél látható lesz, hogy megoldásunkban csak az elengedhetetlenül szükséges szenzorokat használtuk, így nem voltak központi vagy fedélzeti kameráink, csak egyszerű nyomás és iránytű szenzort használtak a robotok.

Azokban az esetekben, amikor a robotcsoport hálózati kommunikációs kényszer mellett dolgozik, a felfedező algoritmusnak kettős feladata van: optimális mozgásokat kell keresnie a felfedezés szempontjából, és ugyanebben az időben biztosítania kell a hálózati kapcsolatokat is. Néhány alkalmazásban, ez a feladat magától értetődik, mivel a robotokon alkalmazott rádió hatósugara nagyobb, mint a felderítendő terület. Az ilyen esetekben a robotcsoport feltételezhetően egy vezeték nélküli, osztott kommunikációs rendszert használ. Számos egyéb munkában ugyanakkor vizsgálandó és megoldandó multi-robot kutatási probléma, ha az egyes robotokon lévő rádió hatósugara sokkal kisebb, mint a felderítendő terület. Ezekben a feladatokban a munkaterületet célszerű cellákra osztani. A robot egy lépését az eredeti és cél cellák határozzák meg. Ez a fajta celledakompozíciós módszer, amikor is a dekompozíció egy szabályos osztástávolságú rácsból áll, könnyebbé teszi a feladat matematikai jellemzését is. A cellaméreteket célszerű a robot rádió hatósugarát figyelembe véve megtervezni. Ezekben az esetekben a felderítő algoritmusok azt feltételezik, hogy van egy Mobile Ad-hoc Networking (MANET) rendszer, amely kommunikációs csatornát biztosít a csoport bármely két robotja között.

M. N. Rooker és Birk [7] egy olyan felderítő algoritmust használt, ahol a kommunikációs hálózatnak folyamatosnak kellett lennie minden pillanatban. Megvizsgálták azt a fontos esetet is, amikor a csoportnak kommunikációs kapcsolatban kellett lennie egy fix bázisállomással, így a kutatási tartomány korlátozott volt a térben. Itt egy idő-lépésben az egész robotcsoport mozgott, és egy hasznosság függvény módszert használtak a legjobb csoport mozgás meghatározásához. Azok a csoportos mozgások, amelyek következtében a kommunikációs hálózat felbomolhatna, olyan összegű negatív költségpontokat kapott, hogy a csoport sosem választotta ezeket a mozgásokat. Az általunk készített algoritmus

ehhez az eljáráshoz hasonlít a legjobban, ezért tesztjeinkben algoritmusunkat ezzel az eljárással hasonlítottuk össze.

A legtöbb felsorolt rendszer feltételezi a vezeték nélküli kommunikáció meglétét. Azokban az esetekben, amikor egy speciális wireless rendszert alkalmaznak a mobil robotok általában WI-FI (IEEE 802.11), Zig-Bee (IEEE802.15.4) vagy Bluetooth (IEEE 802.15.1) rádió rendszerekkel vannak kiegészítve, mivel ezek olcsóak, de mégis kielégítő megoldást nyújtanak.

Sohrabi és társai [8] valamint később Leopold és társai [9] egy új és egyszerű megoldást ajánlottak, miszerint egy egyszerű autonóm egységet (host) kiegészítettek két bluetooth rádióval, így egy nagyméretű wireless szenzor hálózatot alakítottak ki. Egy előző írásunkban [10] a mobil robotok kommunikációs hálózatának a fent leírt dual–rádió rendszerét adaptáltuk a valóságban a Lego cég által gyártott mikrovezérlő alapú NXT modell-robotokra. A dual-rádiós rendszer legfőbb előnye, hogy két kommunikáló robotnak nem kell megszakítania bluetooth kapcsolatát, amíg azok megfelelő távolságra vannak egymástól. Ez azt jelenti, hogy a feladat végzése alatt a kommunikációs hálózat nem egy MANET hálózatként működik, hanem egy statikus hálózatként. Egyrészről ez korlátozza a felfedező algoritmus alkalmazását, másrészről azonban kiaknázhatjuk a dual-rádió alapú rendszer előnyeit, azaz a folyamatosan összekapcsolt hálózat előnyeit, növelhetjük a robotrendszerünk megbízhatóságát és robosztusságát.

2. A területfelderítő algoritmus

A következő részben egy olyan eljárást ajánlunk, amelynek a feladata adott számú robot segítségével a lehető legnagyobb terület felfedezése úgy, hogy a feladat végrehajtása során a robotok egy kommunikációs láncot alkotnak olyan módon, hogy a lánc egy pillanatra sem szakadhat meg. Kényszerfeltétel az is, hogy a lánc egyik vége rögzített, a felderítés alatt végig kommunikációs kapcsolatban áll egy bázisállomással. Egy cellát akkor tekintünk felderítettnek, ha a robotcsoport egy tagja belépett a cellába és elfoglalta a cella középpontját.

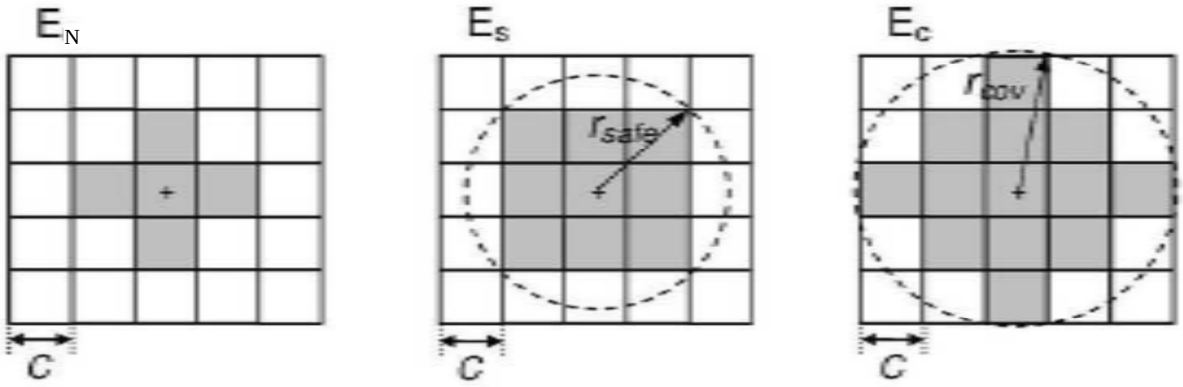
2.1 A cella –méret és a rádió hatótávolság

A felderítendő területet azonos méretű, átfedés mentes, négyzetrács alakú cellákra célszerű felosztani, mint az gyakori a hasonló problémák esetében. Ezekben a feladatokban, a legfontosabb paraméterek egyike a roboton lévő rádió hatósugara. A kommunikációhoz mi bluetooth kommunikációt használunk így a cellák mérete a bluetooth hatósugarától függ. Ezt r_{cov} -nak jelöltük és azt feltételeztük, hogy a rádió kapcsolat ezen a hatósugáron belül minden esetben garantált. Számos cikkben több kiegészítő környezetet is definiálnak az r_{cov}-on kívül. Ezt a környezetet „biztonsági" környezet néven említik és r_{safe} – ként jelölik. A sugár ebben az esetben kisebb, mint az r_{cov}.. Az algoritmus megpróbálja az összes robotot a szomszédjuk biztonsági környezetében tartani, mely nagymértékben megnöveli a csoport robosztusságát kapcsolódási szempontból. Esetünkben bizonyos szituációkban megtörténhet, hogy egy robot elhagyja a szomszédja biztonsági környezetét, ha egy előre nem látható akadály gátolja a tervezett lépést, de az algoritmus garantálja, hogy egyetlen robot sem hagyhatja el a szomszédja rádió hatósugarát normál működés során.

Először megadjuk az r_{cov} és az r_{safe} aktuális paraméter értékeit, ami egyben meghatározza egy cella négyzet oldalhosszát. Ezután az aktuális bluetooth hatótávolság sugarát alapul véve a C speciális értékét határozzuk meg. Az alábbi módon határoztuk meg a két környezeti paraméter értékét:

$$r_{safe} = \frac{3}{2}\sqrt{2}C \quad \text{és} \quad r_{\text{cov}} = \frac{\sqrt{26}}{2}C\,.$$

Egy cella pozícióját a területen jelölje (**P**), amit a cellaközéppont Cartesian-koordinátáival adunk meg. A fent megadott sugarak alapján az egyes cellák **P** környezeteit E_S(**P**)-vel és E_C(**P**)-vel jelöltük, mivel a cellák egy csoportja teljesen belül esik egy r_{cov} és $r_{safe\ sugarú}$ P középpontú körön, mint ahogy ez a *1. ábrán* is jól látható. Ezek mellett E_N(**P**)-ként definiáltuk a **P** környezetét, amit a **P** cellának és az összes vele szomszédos oldalú cellának az uniójaként kapunk.



1. ábra. A robotok környezetei az algoritmusban

Ezek a környezetek nagyon fontosak a bemutatandó felderítő algoritmusban. Tehát, E_C(**P**) –vel jelöljük azon cellák csoportját, ahol a vezeték nélküli kapcsolat garantált egy **P**-ben elhelyezkedő robottal. E_S(**P**)–vel jelöljük azon cellák csoportját, ahol a **P**-ben lévő robot kommunikációs szomszédjai vannak tervezve, egy csoport mozgás befejezése után. Végül jelöljük E_N(**P**) –vel azon cellák csoportját ahová a **P**-ben lévő robot léphet egy csapat mozgás alatt. Ez azt jelenti, hogy egy robot egy csoportmozgás alatt átlósan nem léphet, és nem mozdulhat távolabb a szomszédos cellánál (egy robot helyben is maradhat).

2.2 A kommunikációs lánc

Rendszerünkben a legegyszerűbb topológiájú kommunikációs hálózatot választottuk: egy lineáris láncot (elágazások és ismétlések nélkül). A topológia statikus, azaz a kialakított rádió kapcsolatokat nem lehet megszakítani és nem lehet új kapcsolatokat kiépíteni. A kommunikációs rendszer egészen egyszerű, a kommunikációs lánc egyik vége fixen, egy cella közepén helyezkedik el. Ez egy fix helyzetű bázis állomás, valamint minden robotnak két kommunikációs szomszédja van, az utolsót kivéve.

2.3 A fal-szerű akadályok

A felfedezendő terület tartalmazhat ismeretlen, fal-szerű akadályokat. Egy ilyen akadály meghatározása egyszerű, ha egy robot át tud lépni egy adott cellából a szomszédos cellába, akkor nincs akadály a két cella határvonalán, egyébként a két szomszédos cella közös oldala valószínűleg egy fal-szerű akadály. Egy cellarácsnak bármely szegmense lehet akadály (fal), nincs előzetes ismeretünk az akadályok elhelyezkedéséről vagy sűrűségéről. Az akadály hossza az megegyezik egy cella oldalhosszával és azon két cella határvonalának pozícióival azonosítható, melyek blokkolva vannak az akadály miatt. Ezek az akadályok a kommunikációt nem blokkolják, azaz azt feltételezzük, hogy ha a két szomszédos robot között egy akadály helyezkedik el, akkor az a kommunikációt nem zavarja. Azt feltételezzük, hogy egy akadályt csak akkor fedez fel a robotcsoport bármely tagja, ha megpróbál átmenni az egyik cellából egy vele szomszédos cellába

2.4 A felderítő algoritmus

A robotcsoport tagjait R_i-vel jelöljük, ahol i=1,…N. Az R_i robot kommunikációs szomszédjai az R_{i-1} és R_{i+1} robotok (ha léteznek). R_1 reprezentálja azt a hostot, ami a bázisállomáson van, R_N jelöli azt a robotot, ami legtávolabb van a bázisállomástól a kommunikációs láncban.

A robotok által elfoglalt cellákat vektorok halmazával adjuk meg {$\mathbf{P}_1$,…$\mathbf{P}_N$}. $\mathbf{P}_{1=}$(0,0), ez a bázis állomás pozíciója.

Minden egyes csapatpozícióhoz {$\mathbf{P}_1$,…$\mathbf{P}_N$}. az algoritmus meghatározza a következő csapatpozíciót {$\mathbf{P'}_1$,…$\mathbf{P'}_N$}. Következésképpen az R_i robot egy elemi mozgását a ($\mathbf{P'}_i$-$\mathbf{P}_i$) vektor adja.

Az elkészített területfelderítő algoritmusnak garantálnia kell a lineáris kommunikációs lánc összekapcsoltságát, valamint a lehető legjobb felfedező mozgásokat keresi. Az ajánlott algoritmikus rendszer a robotcsoportot irányító (vezető) és követő robotokra osztja, a következőek szerint:

Az R_N robot a robotcsoport vezetője. Ez azt jelenti, hogy egy lépésciklusban az R_N robot egy alapvető lépését határozzuk meg először. Hasonlóképpen a többi fentebb idézett eljáráshoz, a számítás alapkoncepciója a lehetséges mozgások költsége, amelyeket az R_N relatív pozíciója és a határos cellák határoznak meg.

A többi robot (R_1,…R_{N-1}) mozgását úgy határozzuk meg, hogy az összekapcsoltság biztosítva legyen. Nevezzük őket követő robotoknak. A követő robotok feladata nem csak a kommunikáció biztosítása, hanem felderíthetnek még olyan cellákat is, ahol azelőtt még nem járt robot. Ezt a „másodlagos" feladatot automatikusan végzik az „elsődleges" követő mozgás közben. Az R_N robot felderítő mozgását tulajdonképpen a követő robotok lehetséges mozgásai határozzák meg, ha egy felderítő mozgást nem lehet követni, akkor ez a mozgás tiltott lesz az R_N robot számára.

A csoport következő pozíciója rekurzív úton határozható meg. Először az R_N robot lehetséges új pozícióinak halmazát határozzuk meg, és prioritási sorrendbe rendezzük őket. Nevezzük a pozíciók rendezett halmazát Priority List of New Positions-nak és jelöljük PLN_N –el(magáról a prioritásról majd később esik szó). Ez után az algoritmus megvizsgálja a PLN_N első elemét abból a szempontból, hogy vajon az R_{N-1} követni tudja-e ezt a lépést. Az R_{N-1} lehetséges lépései határozzák meg PLN_{N-1} elemeit. Ha ez üres marad, a $\mathbf{P'}_N$-P_N mozgás nem lehetséges és a PLN_N következő elemének megvizsgálása történik ugyanezen az úton. Ha a PLN_{N-1} nem üres, akkor ugyanez történik, azaz PLN_{N-1} első elemét vizsgáljuk meg, úgy hogy az elfogadható legyen R_{N-2}-nek. Ha nem elfogadható, akkor az eljárás a PLN_{N-1} következő elemét vizsgálja. Így ugyanaz az eljárás ismétlődik minden egyes szinten N-től 1-ig, ahol az i. szint felel a PLN_i elemeinek meghatározásáért. Ez a rekurzív eljárás addig folytatódik, amíg egy elfogadható új pozíciót talál a PLN_N-ben, ami azt is jelenti, hogy egy elfogadható, új pozíciót talált minden robotnak. Ez az eljárás azt is biztosítja, hogy a lehetséges legjobb csoportmozgást találja meg, feltéve, hogy valamennyi PLN_i optimálisan van rendezve.

Az összes, tervezett csoport-pozíció meghatározását az alábbi rekurzív függvény adja:

Team_New_Position(i, [P'_{i+1}…P'_N]) =

$$= \begin{cases} [\ P'_i,\ \text{Team\_New\_Position}(i-1,\ [P'_i \ldots P'_N])\ ] \text{ or } [] & \text{if } i>1 \\ [(0,0)] \text{ or } [] & \text{if } i=1 \end{cases}$$

ahol [P'_{i+1}…P'_N] a tervezett pozíciói az R_{i+1} … R_N robotoknak és P'_i az első pozíció a PLN_i-ben, amelyekben a Team_New_Position(i, [P'_{i+1}…P'_N]) függvényhívás egy nem üres listát ad vissza. Ha nincs ilyen elem a PLN_i-ben, akkor a függvény is egy üres listát ad vissza. Tehát, a Team_New_Position(i, […]) függvényhívás az i-nek megfelelően mindig visszaad egy pozíció listát, és ez a lista egy új pozícióval növekszik minden rekurzív körben. Látható, hogy a Team_New_Position(N-1, [P'_N]) függvényhívás egy teljes, tervezett követő csoport-pozíciót ad vissza [P'_1…P'_{N-1}] ha ez kimenete a P'_N-nek, egyébként egy üres listát kapunk.

A rekurzív Team_New_Position() függvény algoritmusát **Algoritmus 1.** néven jelöltük. Van még egy kiegészítő elem ebben az algoritmusban, azért, hogy a függvény az új pozíciók egy nem üres listájával térjen vissza, a feltétel:

$$PLN(j) \notin \{P'_2, \ldots P'_{i-1}, P'_{i+1}, \ldots P'_N\} \setminus \{(0,0)\}$$

igaz kell, hogy legyen. Ez azt jelenti, hogy a lehetséges új pozíciót PLN(j) csak akkor lehet választani, ha az nem esik egybe a követő robotok többi lehetséges pozícióival, kivéve a bázisállomást (0,0). Mivel ez a feltétel minden szinten vizsgálatra kerül, az összes tervezett új csoport-pozíció rendelkezni fog ezzel a tulajdonsággal, azaz nincs olyan új robotpozíció, ami egybeesik bármely másikkal, kivéve a bázis állomást. Ez a tulajdonság rendkívül fontos az ütközés elkerülése szempontjából, mert ez gyakorlatilag a robotokat egy cella-méret távolságra tartja a többi robottól. Konkrétabban, két robot csak akkor érkezhet így ugyanabba a cellába, ha az egyik közülük egy új akadályt érzékelne, és nem tudná végrehajtani a tervezett lépését. Nem érkezhet három vagy annál több robot ugyanabba a cellába

a bázis állomás (0,0) kivételével, ahol egy sokkal fejlettebb robot navigációs rendszert feltételezünk, mint a területen kívül.

Algoritmus 1:

Team_New_Position (i, [$\mathbf{P'}_{i+1}$ … $\mathbf{P'}_N$]) rekurzív függvény algoritmusa:

```
PLN : = Priority_List_of_New_Positions(i, P'i+1);
if i>1 then
        {
        for all positions in PLN do
          {
          [ P'1 … P'i-1 ] := Team_New_Position ( i-1, [ PLN(j), P'i+1 … P'N ] );
          if [ P'1…P'i-1 ] ≠ [] and PLN(j) ∉ {P'2 , … P'i-1 , P'i+1 ,… P'N} \ {(0,0)}
          then return [ P'1 … P'i-1 , PLN(j) ];
          }
          return [] (empty list) ;
        }
if i=1 then return PLN (which is a list containing only (0,0) or an empty list);
```

A fő területfelfedező algoritmust az **Algoritmus 2.-ben** írtuk le. Először a PLN_N lista lesz generálva a Priority_List_of_New_Positions(N) függvényhívással, majd az algoritmus megvizsgál minden lehetséges új pozíciót a PLN_N-ben prioritási sorrendben, ha az megfelelő lehet egy teljes csoportmozgáshoz. Ez a Team_New_Position (N-1, [PLN_N(j)]) függvény hívásával készül el. A robotok Map nevű objektuma tartalmazza az összes információt, amit a robotcsoport összegyűjtött a területről (akadályok, felderített cellák, stb.). A vezető robot nem csak a csoport által felismert akadályokat regisztrálja az ő Map-jébe, hanem a „virtuális akadályokat" is, amik fontos szerepet játszanak az egész felderítő algoritmusban. Ezek a „virtuális akadályok" valósakként vannak figyelembe véve a vezető robot MANHATTEN távolság megállapító eljárásában (lásd. később). A „virtuális akadályok" bejegyzése két dologból adódik. Először, ha az R_N robot talál olyan mozgást az ő PLN_N –jében, ami a R_{N-1} robot E_C környezetén kívül szeretné juttatni (jelöljük $E_C(P_{N-1})$), akkor blokkolja azt a mozgást, egy újonnan bejegyzett virtuális akadály és az ajánlott PLN_N – beli P_N pozíciót az R_N robot egy u.n. null mozgással helyettesíti. Másodszor, ha a Team_New_Position(N-1,[PLN_N(j)]) függvény hívás a PLN_N elemeinek egy üres listájával tér vissza, akkor még egy virtuális objektum is bejegyzésre kerül a Map-be P_N és PLN_N(j) pozíciói között. Egy virtuális akadály arra a tényre utal, hogy a lánc-szerű csapat nem tudja elérni az adott cellát egy adott lépés segítségével. Ez a cella ugyanakkor, lehet, hogy elérhető később egy másik irányból. A következő csoportmozgás-tervezési ciklusban az R_N olyan új lehetséges pozícióit, amik blokkolva vannak a virtuális objektumok miatt, nem kell kiválasztani a PLN_N-ben. A virtuális objektumok alkalmazása nélkül a csapat folyamatosan próbálná elérni ezeket jelenleg „nem elérhető" cellákat, és nem vennék észre a reménytelen helyzetet. Le kell szögezni, hogy a virtuális akadályok csak az R_N robotot blokkolják, a többi robotot csak a valós akadályok gátolják. A már felfedezett valós akadályokat figyelembe veszik, mielőtt az R_i robot kiszámítja az ő PLN_i -jét. Egy robot csak akkor fedezhet fel felderítetlen akadályt, amikor a csoport lépés kiszámítása befejeződött, és a „start mozgás" eljárást elindítja az egyedi robotokon. Nem biztos, hogy egy pontosan és jól megtervezett csoportmozgást meg lehet valósítani. Ha egy robot nem tud belépni egy újonnan felfedezett valós akadály miatt a cél cellába, akkor a robot az aktuális cellájában a kiinduló helyére megy vissza és jelezi az akadályt. Ha a Priority_List_of_New_Positions(N) függvényhívás nem állít elő egyetlen felfedező mozgást sem (egy üres listát ad vissza), akkor a mozgás egy „visszalépés" lesz (Backtrack_Team-Move- ként jelöljük). Ez azért szükséges, mert ilyen esetekben a vezető robot teljesen el van zárva virtuális vagy valós akadályok miatt egy felfedezendő területtől és csak a visszalépés a megoldás. Rooker és Birk már alkalmazta ezt a megoldást a [7] cikkben. Abban az esetben, ha már nem lehetséges visszszalépni, ugyanis visszatértünk a kezdő pozícióba, akkor az összes virtuális akadály törlődik, és a felderítés folytatódik.

A mozgási fázis végén a robotok egy broadcast üzenetben elküldik új helyzetüket és a felderített akadályokat.

Algoritmus 2:

A fő területfelfedező algoritmus:

while not *Stop_Condition_Is_True* ***do***
{
PLN_N *: = Priority_List_of_New_Positions(N);*
if PLN_N *= [](empty list)* ***then if*** *Backtrack Team-Move is possible* ***then*** *do*
Backtrack Team-Move;
else *delete virtual obstacle from Map;*
for *all position* $PLN_N(j)$ *in PLN* ***do {***
if $PLN_N(j) \notin EC(\mathbf{P}_{N-1})$ ***then***
{
register a virtual obstacle between the cells $\mathbf{P}_N$ *and* $PLN_N(j)$ *in Map;*
$PLN_N(j) := \mathbf{P}_N$ *(i.e. null move);*
}
$[\ \mathbf{P'}_1 \ldots \mathbf{P'}_{N-1}\ ] :=$ *Team_New_Position (* $N-1$ *,* $[PLN_N(j)]$ *);*
if $[\ \mathbf{P'}_1 \ldots \mathbf{P'}_{N-1}\ ] \neq []$ ***then***
{
$\mathbf{P'}_N = PLN_N(j)$;
Robot R_i *starts move* $(\mathbf{P'}_i - \mathbf{P}_i)$ *(* $i = 1, \ldots N$ *);*
Robot R_i *broadcasts its new position and the discovered obstacles;*
Robot R_i *refreshes its Map;*
break *(****for*** *cycle);*
}
else *register a virtual obstacle between the cells* $\mathbf{P}_N$ *and* $PLN_N(j)$ *in Map ;****}***
}

Az algoritmusok kritikus eleme a fent említett Priority_List_of_New_Positions() függvény, amely megszerkeszti az egyedi robotok PLN_i–jét, ez különbözik az R_N és R_i esetében, ahol i<N. Ez az eljárás felelős a felfedező stratégiáért ha i=N és a követő stratégiáért, ha i<N. Ezt az **Algoritmus 3**.-ban részleteztük. Az eljárás azon lehetséges új pozíciók (**P** jelöli) halmazából indul ki, amelyeket a valós és „virtuális ” akadályok halmaza nem blokkol. Ezután kiszámolja a legrövidebb, akadály elkerülő MANHATTEN „utat” egy felderítetlen cellához minden egyes cellától, amelyek engedélyezett mozgásokkal elérhetőek. (Nyilvánvalóan, legfeljebb négy ilyen cella van). Pontosabban, minden $\mathbf{P_{i,j}} \in$ P kiszámítja az út hosszát

$$dj = \min\{\ MDist(\mathbf{C},\mathbf{P'_{i,j}})\ \mathbf{C} \in \text{Map(unexplored)}\ \}$$

ahol a Manhetten távolságot két cella között (pozícióikat **A** és **B** jelöli) MDist(**A,B**)–vel jelöltük, valamint Map(unexplored) jelöli a felderítetlen cellák halmazát. A Manhetten távolság **A** és **B** között a legrövidebb (akadály elkerülő) útként van definiálva egy négyzetrácson, amely tartalmazza **A**-t és **B**-t csomópontként. Az eljárás $\mathbf{P'}_{i,j}$-ből indulva akkor áll meg, amikor az első felderítetlen területet elérte. Amikor a dj értéke megvan, akkor az új pozíciók listájának prioritási rendje is adódik úgy, hogy a $\mathbf{P'}_{i,j}$ pozíciókhoz tartozó kisebb értékű dj-e megelőzi a magasabb dj-ű értékeket. Ez megfelel a szokásos útvonaltervező stratégiáknak a „határvonal” alapú eljárásokban, vagyis, hogy a leginkább előnyben részesített új pozíció fog legközelebb esni a határvonalhoz. Ha nem talált útvonalat egyetlen felfedezetlen cellához sem, akkor a függvény a PLN egy üres halmazával tér vissza. Az i=N esetben ez az elsődleges szempont a lista felépítésében. Ha több $\mathbf{P'}_{N,j}$ pozícióhoz ugyanaz a dj érték tartozik PLN_N-ben, akkor a nagyobb skaláris szorzatú lépés $(\mathbf{P'_{N,j}} - \mathbf{P_N}) \cdot \mathbf{P_N}$ megelőzi a többit, mert ez az új pozíció jobban növeli a távolságot a bázisállomástól, mint a többi. Ez a másodlagos szempont a lista feltöltésében.

Abban az esetben amikor i<N, ez a függvény a felelős a jó „lánc-megtartásos” mozgások megtalálásáért. Következésképpen, az engedélyezett, lehetséges új pozíciók listája úgy van meghatározva, hogy R_i marad az $E_C(P_{i-1})$ és $E_S(P'_{i+1})$ környezetében, ahol P_{i-1} a jelenleg ismert

pozíciója az R_{i-1}-nek, valamint P'_{i+1} az ajánlott új pozíciója az R_{i+1}-nek. Ez azt jelenti, hogy R_i követi R_{i+1}-et azért, hogy R_{i+1} új pozíciójának a szűkebb (biztonságos) környezetében maradjon, ugyanakkor az R_{i-1} jelenlegi pozíciójának szélesebb (rádió sugarú) környezetében marad. Hasonlóan ehhez, az új lehetséges pozíciók halmaza $E_S(P'_{i+1}) \cap E_N(P_i) \cap E_C(P_{i-1})$ metszetként van megadva. Ez biztosítja, hogy a kommunikációs lánc nem fog megszakadni még akkor sem, ha ismeretlen akadályok blokkolnak néhány mozgást a csoportmozgásban. A követő robotok esetében a felderítés csak másodlagos probléma az optimális követési stratégiához képest. Következésképp a követő robotok lehetséges új pozíciói először a növekvő d_j értékek szerint lesznek rendezve, de ez után ez úgy rendeződik át, hogy azok a pozíciók, amelyek az $E_N(P'_{i+1})$ elemei, megelőzik a többit. Az ötlet ez mögött az, hogy R_i számára a legjobb dolog, ha nem csak $E_S(P'_{i+1})$ környezetében lesz, hanem az $E_N(P'_{i+1})$-ben is (P_{high} halmaza), így az R_{i+1} robotnak nagyobb variációs lehetősége lesz a mozgások szempontjából a következő lépésben. Ha P_i pozíció eleme a P_{high}-nak, akkor az optimális új pozíció mozgás nélkül érhető el, így a legjobb akció a helyben maradás. Ebben az esetben ez a pozíció a PLNi-ben az első helyre kerül.

Algoritmus 3:

A Priority_List_of_New_Positions(i) függvény algoritmusa (i=N esetben)
A Priority_List_of_New_Positions(i, $\mathbf{P'_{i+1}}$)) függvény algoritmusa (i<N)
Determine the set of enabled new positions:
P := $E_N(P_N)$ in the case of i=N;
P := $E_S(P'_{i+1}) \cap E_N(P_i) \cap E_C(P_{i-1})$ in the case of i<N;
Delete all positions from P that are blocked by known obstacle;
Virtual obstacles apply only to the case of i=N;

for all $P'_{i,j} \in P$ ***do {***
d_j*:= min{ MDist(***C***,* $\mathbf{P'_{i,j}}$*) | ***C*** is unexplored};*

if *there is no Manhattan path to unexplored cell* ***then return*** *[];*
Line up the $P'_{i,j}$ elements into PLN by increasing d_j value;
if *i=N* ***then*** *{*
Line up the $\mathbf{P'_{N,j}}$ *elements with the same d_j value by decreasing ($P'_{N,j}$ - P_N)·P_N value;}*
if *i<N* ***then*** *{*
P_{high} *:= P∩E_N(*$\mathbf{P'_{i+1}}$*); (set of higher priority elements)*
Put the elements of P_{high} in the first places in PLN;
if $P_i \in P_{high}$ ***then*** *put P_i in the first place in PLN;*
}
return *PLN;*

2.5 A stop feltétel

A felfedezés megáll, ha a stop_condition az Algoritmus 1.-ben true (igaz) értékké válik. Nyilvánvalóan akkor kellene true-nak lennie ennek a feltételnek, ha az összes cella felfedezett lesz. Ez nem teljesíthető minden tetszőleges akadály konfiguráció esetében, még ha a terület kisebb is, mint a robot-lánc hatósugara. Ezért egy másik stop feltételt is be kell építeni az algoritmusba. A felfedezés időtartama az egyedi robotok akkumulátorainak működési időtartamai miatt is korlátozott. Az elektromos energia nagyobb részét a mozgások emésztik fel, egy kisebb részt a rádió és a használt szenzorok. Az utóbbi rész megegyezik nagyjából minden roboton. Ezt alapul véve azt feltételeztük, hogy az indulásnál teljesen feltöltött akkumulátorok minden robot esetében ugyanannyi korlátozott számú mozgást tesznek lehetővé. Ezt a számot MAX_MOVES –val jelöljük. A felderítés időtartamát a következő feltétellel korlátozzuk:

$$\max(\text{MOVES}(R_i)) <= \text{MAX\_MOVES}$$

ahol MOVES(R_i)- vel jelöljük a R_i robot összes mozgásainak számát. Például a MAX_MOVES a Dual NXT robotok esetében hozzávetőlegesen 400.

2.6 Helyreállítás a robot hibák esetén

A robot-meghibásodások esetén a fő cél, hogy a csoport folytatni tudja a felfedezést, miután a csoport egy tagja leszakad a motor, vagy a kommunikációs lánc hibája miatt. A „leszakadás" alatt azt értjük, hogy a robot már nem vehet rész a felfedező mozgásokban vagy a csoport kommunikációjában. Bármelyik esetben a kommunikációs lánc is sérül, mert ha a robot nem tudja követni a többi robot kollektív mozgását, akkor előbb-utóbb a szomszédjai rádió hatósugarán kívülre kerül. Ezért egy leszakadást minden esetben, a kommunikációs lánc hibájaként veszünk figyelembe. Mivel minden robot képes arra, hogy a vele kapcsolatban álló szomszédja helyzetét észlelje, azaz a csoport tisztában van a hibás robot azonosítóival. Ezekre tekintettel a javasolt helyreállítási eljárást az **Algoritmus 4.**-ben részleteztük. Az alap ötlet nagyon egyszerű, a hibaponton túl lévő minden robot alacsonyabb sorszámú szomszédos robotjának pozíciójához mozog, majd ezután a robotok újraszervezik a kommunikációs láncot, kihagyva a hibáért felelős robotot. Ez után a felderítést tovább lehet folytatni egy rövidebb lánccal. Ez a helyreállító eljárás csak akkor működik, ha az összes robot ismeri a többi robot aktuális helyzetét, amelyet az **Algoritmus 2**. biztosít.

Egy speciális eset, amikor a vezető R_N robot hibásodik meg. Eddig nem kezeltük azt a kérdést, hogy az algoritmikus számítások (csoport mozgások) fizikailag hol történjenek. Valójában ezek a számítások valamennyi roboton teljesülhetnek (beleértve az R_1 bázisállomást is), mert ezek nem igényelnek magas számítógép kapacitást. Ezt figyelembe véve, a vezető R_N robotot az R_{N-1} –es robot helyettesítheti, ha az R_N leszakad.

Algoritmus 4.

Helyreállító algoritmus egy robot leszakadása esetére:

if *both links of R_j fail* ***then***
{
R_i moves to the position $\boldsymbol{P_{i-1}}$ (the present position of R_{i-1}) ***for*** *i=(j+1), … N;*
Establish the link between R_{j-1} and R_{j+1} (continue exploration without R_j);
}
if *the link between R_j and R_{j+1} fails* ***then***
{
R_i moves to the position P_{i-1} (the present position of R_{i-1}) ***for*** *i=(j+1), … N;*
if *1 < j < (N-1)* ***then***
{
Establish the link between R_{j-1} and R_{j+1} (continue exploration without R_j);
if *it is not successful* ***then*** *Establish the link between R_j and R_{j+2} ;*
(continue exploration without R_{j+1});
}
if *j = 1* ***then***
{
Establish the link between R_1 and R_3 (continue exploration without R_2);
if *it is not successful* ***then*** *R_2 will be the base station instead of R_1 ;*
}
if *j = (N-1)* ***then***
{
Establish the link between R_{N-2} and R_N (continue exploration without R_{N-1});
if *it is not successful* ***then*** *R_{N-1} will be the leader robot instead of R_N ;*
}
}

3. Az algoritmus tesztjei

3.1 Az algoritmus megvalósíthatósága

A bemutatott algoritmus valós robotcsoportra való alkalmazásához nem szükségesek magasan kvalifikált robotok. Általános esetekben a robotcsoport tagjait a következő tulajdonságokkal kell ellátni (az előbb bemutatott kommunikációs képességeken túl):

- Navigációs képesség, a pontosság érdekében legalább 3.00*E-3.
- A robotok pozícióinak pontos értékekkel való frissítési képessége, ha azok a bázis állomás cellájában vannak.
- Helyi akadály meghatározó képesség. Ez azt jelenti, hogy a robot képes észlelni egy akadályt egy távolságban (ez a távolság az ütközésérzékelőt használó robotok esetén 0).
- Önállóan dolgozik rajta egy kis-akadály és társ robot elkerülő algoritmus.

A meghatározás első pontja szerint, ha egy 3.0*E-3 navigációs képességű felderítő robot keresztülhalad **n** cellán (mérete C) akkor a pozíciójának hibája (3.0*E-3)*n*C lehet. Ennél fogva, ha egy robot 333 cellán halad keresztül, akkor a hibaérték nagysága egy cella oldalmérete lehet. A következő fejezetben egy 15x15 cellából álló szimulációs területen teszteltük az ajánlott algoritmust Dual-NXT robotok segítségével is. Ezen a területen a leghosszabb felfedező út hossza 100 cella alatt van, ha az akadálysűrűség kicsi. Ilyenkor az sem jelent túl nagy problémát, ha a fordulások hibái miatt a robotok navigációs pontossága romlik. Magasabb objektumsűrűség esetében ez az adat 100 - 400 cella, de a robotok többször is „visszatérhetnek a bázis állomásra, így a pozíciók frissülnek Nagyobb teszt-területen a robotcsoportnak irányítottan kellene visszatérnie a bázis állomásra annak érdekében, hogy frissítsék saját pozíciójukat (vagy pontosabb navigáció kell).

Mint arról már egy előző cikkben [11] is írtunk, egy szenzorfúziós eljárás segítségével differenciál két kerék meghajtású Dual NXT (tribot) robotokkal, ahol szinkronizált kerekeket és iránytűt is használtam a pontos navigáció érdekében, 3.00*E-3 pontosságot értem el egyenes szakaszokon. A helyi akadály-elkerülés még viszonylag egyszerű eszközökkel is megvalósítható. Meg lehet oldani ütközés, távolság vagy lézer szenzorok segítségével. Mi csak ütközés érzékelőt használtunk. Az ajánlott algoritmusunkat Dual NXT robotokkal teszteltük. Ha a robot egy akadályba ütközött, akkor 30 cm hátrált, derékszögben jobbra fordult, előre ment 30 cm-t, majd balra fordult 90 fokot. Ezután, ha az előre haladáskor újra ütközött, akkor az akadály egy fal, különben egy társ robot.

3.2 Az algoritmus optimális akadálymentes esetben

Mi akkor nevezzük „optimálisnak” az algoritmust, ha az felfedezi az összes elérhető cellát az adott kényszer feltételek mellett, a legkisebb időráfordítással. A felfedezési időt, időegységekben mérjük, ahol egy időegység az az idő, ami alatt a robot az egyik cellából átmozog egy vele szomszédos cellába.

Azt állítjuk, hogy az eljárásunk az adott feltételekkel optimális akadálymentes esetben, azaz a lehető legnagyobb területet fedezi fel a legkisebb idő alatt.

Az állítás bizonyítása érdekében megmutatjuk, hogy ha nincs akadály a területen, akkor a felfedezési idő nem lesz több 9*(N-1) időegységnél, N robottal, valamint a robotok felfedezik az összes elérhető cellát. Nevezzünk egy cellát „határ cellának”, ha az elérhető a csoport által, de az ő oldal szomszéd cellája viszont nem érhető el. Esetünkben 8*(N-1) határ cella van és ezek egy 2*N-1-szer 2*N-1 oldalú négyzet mentén helyezkednek el. Kettő vagy több robot nem foglalhat el határ cellát ugyanabban az időlépésben, mivel a bázis állomáshoz kapcsolódás feltétele nem lenne biztonságos. Ezért csak egy új határ cella fedezhető fel egy időlépésben, ami azt jelenti, hogy a határ cellák sorozatának felderítése legalább 8*(N-1) időegység. A legrövidebb út a bázis állomástól a legközelebbi határcellához N-2 cella hosszúságú, ezért, legalább 8*(N-1)+N-2=9*(N-1)-1 időegység szükséges az összes cella felderítéséhez.

Második lépésként megmutatjuk, hogy az algoritmus 9*(N-1)-1-ben valósítja meg a felfedezést. Ez az időegységek minimális száma. Az első fázisban a vezető robot elmegy a legrövidebb úton a legközelebbi határ cellához. Az első robot egyenesen halad előre ebben a fázisban, így a PLN_N első

eleme mindig új pozíció lesz, hogy legjobban növelje a bázis állomástól való távolságot, azaz nagyobb a skalár szorzat $(P'_{N,j} - P_N)\cdot P_N$. Ennek a fázisnak N-2 időegységre van szüksége. A második fázisban a vezető robot végigsétál a határ cellákon egymás után, mert a $PLN_{N\ -ben}$ az első érték (a nem elérhető pozíciók után) mindig a következő határcella lesz. Ez ugyanaz, mint a legközelebbi elérhető felfedezetlen cella. Amíg a vezetőnek ez a mozgása, a követő robotok formája egy mozgó vonal a bázis állomás és a vezető robot között, így felfedeznek minden cellát a bázis állomás és a határcella között. Mivel nincsenek akadályok a területen, ez a mozgásminta megvalósítható. A robotok felfedező útjai a második fázisban koncentrikus négyzeteken fekszenek a bázis állomással a központjukban. A második fázisnak 8(N-1) időegységre van szüksége, ami a határcellák száma. Az egész felfedezés 9(N-1)-1 időegységet tesz ki.

3.3 Az algoritmus összehasonítása referencia algoritmusokkal

Az elkészített rekurzív lánc algoritmusunk teljesítményét összehasonlítottuk a Rooker és Birk [7] által publikált algoritmussal. Eljárásukat „referencia eljárásnak" neveztük el. Algoritmusuk egy költség függvényt használt annak érdekében, hogy kiválassza a legjobb csoport-mozgást a lehetséges csoport mozgások viszonylag nagy halmazából, melyek véletlenül voltak generálva.

Alapvető különbség a két algoritmus között, hogy a referencia algoritmus egy költség függvényt használ, ami azonos súllyal veszi figyelembe az egyedi robotlépések költségét, valamint a robotok szerepei azonosak, így a felfedező feladat és a költségfüggvény decentralizált. Ezen kívül, a referencia eljárás egy folyamatosan működő MANET rendszert feltételez, míg az általunk ajánlott eljárás nem. A másik jelentős különbség, hogy a referencia eljárás csoport lépés tervező függvénye magas számítási igényeket foglal magába, míg a miénk sokkal kevesebb számítást igényel.

Ezen kívül alkalmaztunk egy másik referencia algoritmust is. Ez egy speciális esete Rookerék algoritmusának, ahol a legjobb költségű csoport-mozgás az összes, lehetséges csoport-mozgás közül van kiválasztva. Ez a készlet 5^{N-1} csoport-konfigurációt tartalmaz, ami a jelenlegi tesztünkben, ahol N=8, $5^7 \approx$ 78125 lehetőség. Nevezzük ezt „Teljes keresési" algoritmusnak. Ez az algoritmus rendelkezik a lehetséges legjobb csoportmozgással, amit a költségfüggvény ad [7]. Ténylegesen ez az algoritmus csak elméleti jelentőségű, mert a csoportmozgások számítása a kb. 78125 csoport konfiguráció hasznosság függvényének értékelését is tartalmazza, amely több mint 78- szor hosszabb, mint a fent említett referencia algoritmus.

Ez után a három algoritmust, MATLAB-ot használva, számítógép szimulációval teszteltük N=8 robottal, ami tulajdonképpen 7 felderítő robotot jelentett. Kétféle akadálytípust használtunk a szimulációs területeken. Az egyik konfiguráció *p* akadálysűrűségű, véletlen eloszlású, akadályokat tartalmazott, azaz bármelyik cellaelválasztó fal p valószínűségű volt, az akadályok közötti bármilyen korreláció nélkül. A másik tesztterület hasonló volt a Rooker és Birk által használt területhez, amely egy épületszerű környezet volt kisebb- nagyobb szobákkal, egyenes falakkal és nyitott ajtókkal szétválasztva. *(1. táblázat)*

1. táblázat Az algoritmusok összehasonlítása

	Akadálymentes terület		Épületszerű terület nagy szobákkal	Épületszerű terület kicsi szobákkal
	Sikeres felderítési arány (%)	Felderítési idő	Sikeres felderítési arány (%)	Sikeres felderítésiarány (%)
Rooker algoritmus	100	372	25	43
Teljes keresés algor.	100	117	67	84
Ajánlott algor.	100	62	75	77

Akadálymentes területen a robotcsoport az összes, 225 cellát felfedezheti, így mérhetjük a felfedezési időt 100% felfedezési arány esetén. A *1. táblázat* mutatja ezt a mérési időt az első két adatoszlopban. Az ajánlott algoritmus előnye ebben az esetben a leghangsúlyozottabb. A felfedezési ideje kb. fele a „Teljes keresési" eljárásnak és kb. egy hatod része a referencia algoritmusnak. A táblából még

leolvasható a felderítés sikeres aránya %-ban megadva, épület-szerű akadálykonfigurációk esetén. Ezekben a szimulációkban mindhárom algoritmus 100%-os felderítés előtt megállt, mert a robotok egyike elérte a MAX_MOVES = 400–as értéket. Ebben a részben a robotok megállítása előtti sikeresen felderített területek arányát hasonlíthatjuk össze. Az ajánlott és a „Teljes keresési" eljárás esetében hasonló volt a felderített területek aránya mindkét konfiguráció esetében (kb. 70% és 80%). Az első esetben az ajánlott a jobb, míg a másodikban a második algoritmus ért el valamivel jobb eredményt. Emellett mindkét algoritmus sokkal jobb eredményt ért el ebben a mérésben, mint a referencia algoritmus, ami csak 25%-os és 43%-os sikeres arányt teljesített.

4. Összefoglalás, távlati célok

Az ajánlott algoritmus és az implementált kommunikációs lánc tesztjeinek eredményét alapul véve a következő következtetéseket vonhatjuk le:

Az általunk készített (fix, lánc-szerű csoport) felfedező eljárás optimális az akadálymentes esetben, a bázis állomással való kapcsolat megtartásának kényszere mellett.

Az általunk készített algoritmus jobb felderítési időt ért el alacsony akadálysűrűségeknél, valamint 75%-os vagy 100%-os felderítési arány esetében, mint a referencia eljárás (decentralizált költség függvény és MANET alapú). Az eljárás kedvező tulajdonsága a még nem akadálymentes területen is annak köszönhető, hogy a vezető robot lépésköltsége felülbírálja a többi robotét a területfelderítő stratégiában. Ez az eljárás elkerüli azt a hatást, hogy a csoport tagjainak egyedi felderítő céljai akadályozzák egymást, amely a kapcsolat fenntartás kényszerű, decentralizált, költség függvény alapú eljárások jellemzője. Le kell szögezni azonban, hogy jó tulajdonságai is vannak az említett decentralizált eljárásoknak, robosztusabbak és gyorsabbak lehetnek olyan területen ahol sok a véletlen akadály. Ezért jövőbeli munkánk tárgya lehet egy olyan hibrid eljárás, amely mindkét típusú eljárás előnyeit tartalmazza.

Az algoritmusba épített ütközéselkerülési stratégia segíti a felderítés speciális alkalmazását szimulátorban vagy egy valós környezetben, ugyanis amíg elméletben a robotok pontszerű egyednek tekinthetőek, gyakorlatban valós kiterjedéssel rendelkeznek (a Dual NXT 22x18x14 cm).

Az algoritmusba épített helyreállító eljárás lehetővé teszi, hogy robot-meghibásodás esetén is folytathassa a felderítést a robotcsoport. Az eljárás külön kezeli, ha a lánc végén lévő robot a lánc közepén lévő robot vagy a bázisállomáson lévő robot hibásodik meg.

Az általunk ajánlott algoritmust implementáltuk valós Dual NXT robotokra is, és teszteltük akadálymentes területen. A robotokon működött az ütközéselkerülő **e**ljárás, valamint szakadás esetére a láncújraépítő eljárás is.

5. Köszönetnyilvánítás

A tanulmány írása „a TÁMOP 4.2.4.A/2-11-1-2012-0001 azonosító számú Nemzeti Kiválóság Program – Hazai hallgatói, illetve kutatói személyi támogatást biztosító rendszer kidolgozása és működtetése konvergencia program című kiemelt projekt keretében zajlott. A projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg."

IRODALOM

[1] R. Simmons, D. Apfelbaum, W. Burgard, D. Fox, S. Thrun, and H. Younes, Coordination for multi-robot exploration and mapping, In Proceedings of the National Conference on Artifcial Intelligence AAAI, pp. 852-858, 2000.

[2] W. Burgard, M. Moors, C. Stachniss, Coordinated multi-robot exploration, IEEE Transactions on Robotics, 21(3), pp. 376–378, 2005.

[3] W. Sheng, Q. Yang, J. Tan N. Xi, Distributed multi-robot coordination in area exploration, Robotics and Autonomous Systems 54,pp. 945-955,2006.

[4] J. Vazquez, C. Malcolm, Distributed multirobot exploration maintaining a mobile network, IEEE International Conference on Intelligent Systems pp. 113–118, 2004.

[5] Y. Pei, M. W. Mutka and N. Xi, Coordinated multi-robot real-time exploration with connectivity and bandwidth awareness, IEEE International CON FERENCE on Robotics and Automation (ICRA), pp. 5460 – 5465, 2010.

[6] T.S. Dahl, M. Mataric, G.S. Sukhatme, Multi-robot task allocation through vacancy chain scheduling, Robotics and Autonomous Systems 57 (6–7) 674–687, 2009.

[7] M.N. Rooker, A. Birk, Multi-robot exploration under the constraints of wireless networking, Control Engineering Practice 15, pp.435–445, 2007.

[8] K. Sohrabi, W. Merrill, J. Elson, L. Girod, F. Newberg, W. Kaiser, Scalable self-assembly for ad hoc wireless sensor networks, IEEE Transactions on Mobile Computing , pp.317-331, 2002.

[9] M. Leopold, M. B. Dydensborg, P. Bonnet, Bluetooth and sensor networks: A reality check, in Proc. 1st ACM Conference on Embedded Networked Sensor Systems (SenSys), pp. 103–113, 2003.

[10] A. Pasztor, T. Kovács, Z. Istenes, Piconet and Scatternet Communication Networks in Swarm Intelligence Simulation with Mobile Robots Transactions on Automatic Control and Computer Science –SCIENTIFIC BULLETIN of "Politehnica" University of Timisoara 2009/54/3, ISSN-1224-600X, page131- 136.

[11] A. Pásztor, T. Kovács, and Z. Istenes, Compass and Odometry Based Navigation of a Mobile Robot Swarm Equipped by Bluetooth Communication, ICCC CONTI 2010, IEEE International Joint Conferences on Computational Cybernetics and Technical Informatics , 27-29. 05. 2010.