

Lokalizált szenzor önelrendezés optimális fókuszált lefedéshez

Tudományos Diákköri Dolgozat

Szerző: Botka Gábor, 3.féléves MSc hallgató

Témavezető: Dr. Lukovszki Tamás, egyetemi docens

Eötvös Loránd Tudományegyetem

Informatikai kar - Információs rendszerek Tanszék

2012

Tartalomjegyzék

1. Bevezetés.....	3
1.1. Fókuszált lefedettség (F-Coverage) [2].....	4
1.2. Háromszög rács	5
1.3. A probléma ismertetése.....	5
1.4. Korábbi munkák.....	6
1.5. A dolgozat által elért eredmények.....	8
2. Az algoritmus.....	9
2.1. Navigáció a háromszögrácson.....	9
2.2. Az algoritmus ismertetése.....	10
2.2.1. Keringés.....	11
2.2.2. Egyszerű közelítő lépés.....	11
2.2.3. Egyszerű közelítő lépés II. (nincs forgás irányú szomszéd).....	12
2.2.4. Lépés sarokról.....	12
2.2.5. Lépés sarokról a legbelső hatszögön.....	13
2.3. A konvergencia elemzése.....	14
3. Szimulációs környezet.....	17
3.1. Szenzor implementáció.....	17
3.1.1. SensorGraphic osztály.....	17
3.1.2. SensorControl interfész.....	17
3.2. Környezet által biztosított eszközök.....	18
3.2.1. Szenzor műveletek	18
3.2.2. Megjelenítés.....	19
3.2.3. Szimulációs xml bemenet és annak feldolgozása.....	19
3.2.4. Generátorok.....	19
3.2.5. Szimulációk futtatása.....	20
4. Eredmények.....	21
5. További tervek.....	24
6. Összefoglalás.....	25
Irodalom jegyzék.....	26

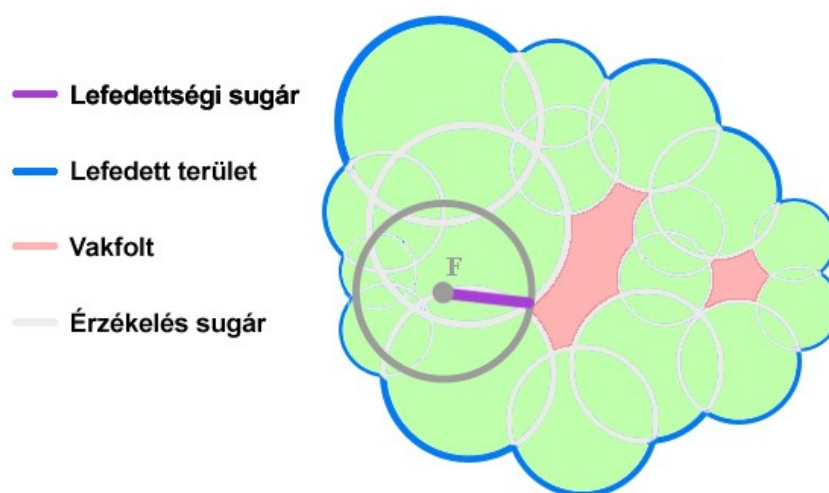
1. Bevezetés

A szenzor hálózatok kialakítása során fontos kutatási területnek minősül, hogy miként tudunk kialakítani egy összefüggő lefedett területet, ha minden egyes érzékelő csak a saját mozgásáért felel. A kutatások sok nyitott kérdésre keresnek még választ, hogyan lehet felkészíteni ezen hálózatokat az esetleges szenzor meghibásodásokra, miként lehet nagyon nagy méretű hálózatokat kialakítani és fenntartani, milyen módon kövessék a szenzorok a környezetben történő esetleges változásokat. A TDK dolgozatomban a hálózat kialakítására vonatkozó algoritmusokat vizsgálom meg, és készítek el egy új megoldást.

A fókuszált lefedés problémája vezeték nélküli szenzor hálózatokban azt jelenti, hogy egy maximális sugarú területet kell lefedni érzékelési vakfoltok nélkül egy kitüntetett célpont (point of interest, POI) körül (lásd [1] [2].) A POI lehet egy vegyipari üzem, amelynek közvetlen környezetében a szenzorok a szennyezettséget monitorozzák, vagy egy katonai alkalmazásban a főhadiszállás, amelynek a védelmét olyan szenzorok segítik amelyek képesek az önálló elrendeződésre. Ezeknek a szenzoroknak lokális információk alapján, centrális irányítás nélkül kell garantálni a vakfolt-mentes lefedettséget. Az [1] és [2] cikkekben bemutatott megoldások optimális fókuszált lefedettséget garantálnak véges idő alatt. A dolgozatban azt vizsgálom, hogy a [3] cikkben bemutatott megoldás egy célpont bekerítésére mobil robotok segítségével hogyan adaptálható a fókuszált lefedés problémájára. A [3] cikkben elméletileg is bizonyított, hogy a mobil egységek $O(D)$ idő alatt képesek bekeríteni a célpontot, ahol D a mobil robotok távolságának összege a probléma kezdeti állapotában. Megmutatom, hogy ezen bizonyítások hogyan használhatóak a fókuszált lefedés problémájára, ahol azt feltételezzük, hogy a szenzorok maximum 2-hop távolságából nyert információira támaszkodnak. A bizonyítás megfelelő átalakítása mellett a fenti elméleti korlát továbbra is érvényes marad. Fontos megjegyezni, hogy ez az elméleti korlát lényegesen alacsonyabb, mint a [1] és [2] cikkben szereplő korlátok. Emellett átfogó szimulációkkal is igazolom az elméleti korlát érvényességét, valamint azt, hogy ez a módszer a gyakorlatban is gyorsabb, mint az [1] és [2] cikkben leírt módszerek.

1.1. Fókuszált lefedettség (F-Coverage) [2]

Lefedett területként definiáljuk az elhelyezett szenzorok csoportjának külső határvonala által zárt terület. Vakfoltnak nevezzük a lefedettségi terület azon részét, amely egyetlen szenzor érzékelési területéhez sem tartozik. Lefedettség mértékét a felület nagyságával jellemezzük. Tehát lefedettség a fedett terület és a vakfoltok területének különbsége. Ideális lefedettség esetén maximalizáltuk a fedett területet anélkül, hogy az vakfoltokat tartalmazna. Fókuszált lefedettség definiálásához nem elegendő a fenti két fogalom ismerete, mivel POI és a legközelebbi fedetlen terület távolsága is fontos tényező. Ezért definiáljuk a lefedettségi sugár fogalmát.

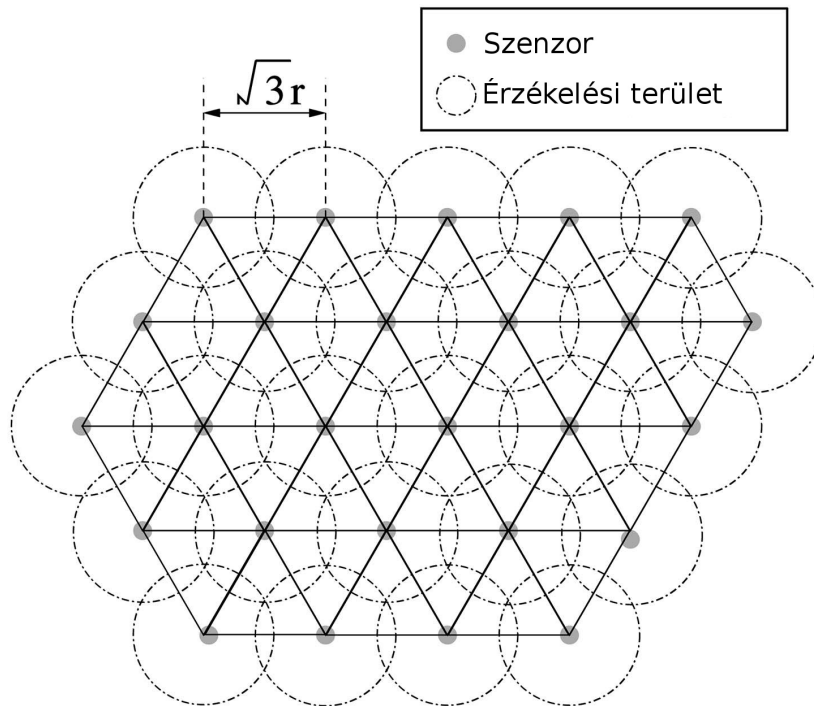


1. ábra: Lefedettségi sugár

Lefedettségi sugár: azon körlap sugara melynek központja a POI és nem tartalmaz fedetlen területet.

Az optimális fókuszált lefedettség maximalizálja a lefedettségi sugarat. Amennyiben végtelen számú szenzor áll rendelkezésünkre és az érzékelési sugár tart nullához, akkor az optimális leszórás tart egy körhöz. Ezért szokás ezt a lefedettségi kör sugarának is nevezni.

1.2. Háromszög rács



2. ábra: Tökéletes szenzor eloszlás [5]

Az [5] cikkben megmutatták, miként lehet Delauney háromszögekkel maximalizált összefüggő lefedettséget biztosítani vakfoltok nélkül, a többszörösön lefedett területek minimalizálása mellett. Ennek feltétele az volt, hogy mindegyik háromszög szabályos legyen és a háromszög oldala $\sqrt{3}r_s$ hosszúságú, ahol r_s az érzékelési kör sugara. Amennyiben a szenzorok kommunikációs sugarát r_c -vel jelöljük és $r_c \geq \sqrt{3}r_s$ a fenti ellenrendezés azt is garantálja, hogy hálózatunk összefüggő lesz.

A [2] cikkben a fenti bizonyítást kiegészítve megmutatták, hogy ha a háromszög rácson egy kitüntetett F pont körül, az F ponthoz a lehető legközelebb helyezkednek el a szenzorok, akkor az F pontra lefedettségi sugár maximális lesz, ezáltal optimális fókuszált lefedettséget kapunk.

1.3. A probléma ismertetése

Adott n darab önálló mozgásra képes szenzor, és egy F célpont amit közre kell fogniuk. A szenzorok egy olyan háromszög rács különböző pontjaiban helyezkednek el, amelynek F pont is része. A szenzorok kommunikációs sugarát r_c -vel, érzékelési sugarát r_s -sel fogjuk jelölni. A feladat, hogy F pontra fókuszált lefedettség maximális legyen.

A feladat megoldása során feltesszük még a következőket:

- 1) A kommunikációs és észlelési sugár között fent áll a következő összefüggés
$$r_c = 2\sqrt{3}r_s \text{ .}$$
- 2) Minden szenzor ismeri saját térbeli koordinátáját, amelynek meghatározása a készülékhez rögzített GPS-szel, vagy valamely más hatékony lokalizációs algoritmussal történik.
- 3) Az F célpont térbeli koordinátája minden szenzor számára ismert.
- 4) Minden szenzornak van egy egyedi azonosítója amit nyomkövető jelként sugároz a kommunikációs sugarán belül. (Ezen kívül más kommunikáció nem megengedett)

1.4. Korábbi munkák

A fókuszált lefedettség fogalmát először [2] cikkben definiálták. Mivel sok más algoritmus született a POI (Point of Interest) bekerítésére amely még nem ismerhette ezen mértéket, így meglehetősen hiányosak az adatok arra vonatkozóan, hogy milyen teljesítménnyel rendelkeznek fókuszált lefedettség terén. Ezért ezt a tulajdonságot mellőzve vesszük végig a kapcsolódó munkákat.

A [6] forrás kiterjedt összefoglaló anyag vezeték nélküli hálózatok terén. Ezen anyag összesen ötféle disztributív vagy lokalizált algoritmikus megközelítést ismertet, hogy miként lehet mobil szenzorhálózatokat építeni.

Az első a vektor alapú (Vector-based) megközelítés, ahol a szenzorok minden körben megpróbálnak olyan mozgás vektort választani, ami garantálja hogy optimális távolságra kerüljenek az őket körülvevő többi szenzortól. Minden szenzor a kiszámított vektor irányába halad amíg el nem éri a kijelölt pozíciót, vagy véget nem ér az adott mozgásra szánt időkeret. Ezen típusú megközelítést felhasználó munkák például a [7] és [8].

Ezen módszerhez állnak talán a legközelebb azon sztochasztikus algoritmusok amelyeknél a szenzorok, a sűrűn lefedett területekről a ritkábban fedettek felé vonulnak, különböző módokon megszorított véletlen bolyongással.[9]

Több a Voronoi sokszögeken alapuló technika is létezik, mint például az [5] és [11]. A működési elvük alapja, hogy mindegyik szenzor a saját sokszögén belül maximalizálni kívánja a fedett területet. Ezt oly módon érik el, hogy igyekeznek olyan pozíciót

felvenni, amelyben az érzékelési környezetük körvonala minél több helyen érintkezik a befoglaló sokszög oldalival.

A negyedik stratégia a terhelés elosztáson (Load balancing) alapul. Itt felosztjuk a teret kisebb részekre és szenzorok képezik a terhelést. Az algoritmus síkidomok mentén kezeli és osztja el a terhelést, fontos kiemelni, hogy ezen megközelítés is lokális adatokat használ csak fel. Ilyen algoritmus a [12].

Az utolsó a [13] és [2] cikk által is használt módszer, hogy a felületre amin a szenzorok mozogni fognak, egy földrajzi gráfot helyezünk és a feladatot átfogalmazzunk ezen gráf pontjainak elfoglalására.

A [2] cikk szerzői a probléma megoldására olyan algoritmust dolgoztak ki amelyben a szenzorok a célpont bekerítéséhez 1-hop távolságból nyert információkra támaszkodnak, de ezen a távolságon belül bármennyi kommunikáció megengedett. Amennyiben a szenzorokat rácspontokról indítottuk, az ő megoldásuk is garantálta, hogy ütközések elkerülése mellett bekerítik az adott célpontot. Ennek elérése érdekében összesen kilenc egymásra épülő szabályt fogalmaztak meg, melyeknek egy része korlátozta a szenzorok forgás irányát illetve, hogy mely rácspontokból megengedett a célpont irányába való haladás. A [2] cikk által megfogalmazott feladat akkor ér véget, amikor a szenzorok bekerítették az adott célpontot, és mozgásukat befejezték, erre vonatkozóan a cikk bizonyítja, hogy ez véges időn belül bekövetkezik.

A [4] cikkben a robotok egy fix vagy lassan mozgó célpontot kerítenek be koncentrikus körpályákon. Egy robot egyik körpályáról a másikra, akkor léphet át amennyiben elég hely áll rendelkezésre, és lokálisan versenyhelyzetben ő élvez elsőbbséget. Az egymást követő körpályák forgásiránya ellentétes, ami jelentősen gyorsítja a konvergencia sebességét. A robotok a bekerítést követően is mozgásban maradnak a célpontot körülvevő pályákon. A cikkben továbbá bizonyították, hogy a mobil egységek $O(D)$ idő alatt képesek bekeríteni a célpontot, ahol D a mobil robotok távolságának összege a probléma kezdeti állapotában.

Mivel TDK során az új adaptált algoritmust [2] és [4] cikkek ötvözeteként állítom elő és az általa elért eredményeket a [2] forrásban található algoritmusok eredményeihez viszonyítom, ezért fontos néhány szót szólni az új algoritmus és annak alapjául szolgáló másik két megoldás főbb különbségeiről. A probléma amit a [2] cikk algoritmus megoldott eltérő kezdőkörülményekkel volt definiálva. Az egyik fő különbség, hogy az ő általuk használt felületen a szenzorok egyenletesen voltak szétszórva, ezáltal első

lépésben a szenzoroknak rácspontokra kellett pozicionálniuk magukat. Szintén az előfeltétel különbségéből adódik, hogy az esetleges ütközéseket feloldják, az algoritmus képes növelni a szenzor POI-tól mért távolságát. Továbbá fontos még azt is megjegyezni, hogy több olyan szabály is van ami feltételezi, hogy tudjuk a közvetlen szomszédok mozgásállapotát vagy a következő mozgásuk irányát. Ezeket a helyzeteket a modell legtöbbször azzal oldja fel, hogy a szenzorok aszinkron különböző sebességekkel mozognak, de így sokszor kénytelenek kommunikálni egymással a szomszédos egyedek. Utolsó említésre méltó különbség ami még eltéréseket eredményez a két modell között, hogy amíg a [2] forrás algoritmus stabil végállapottal rendelkezik, addig [4] cikk megoldásánál és ezért az új algoritmusban is a robotok a célpont bekerítését követően is mozognak.

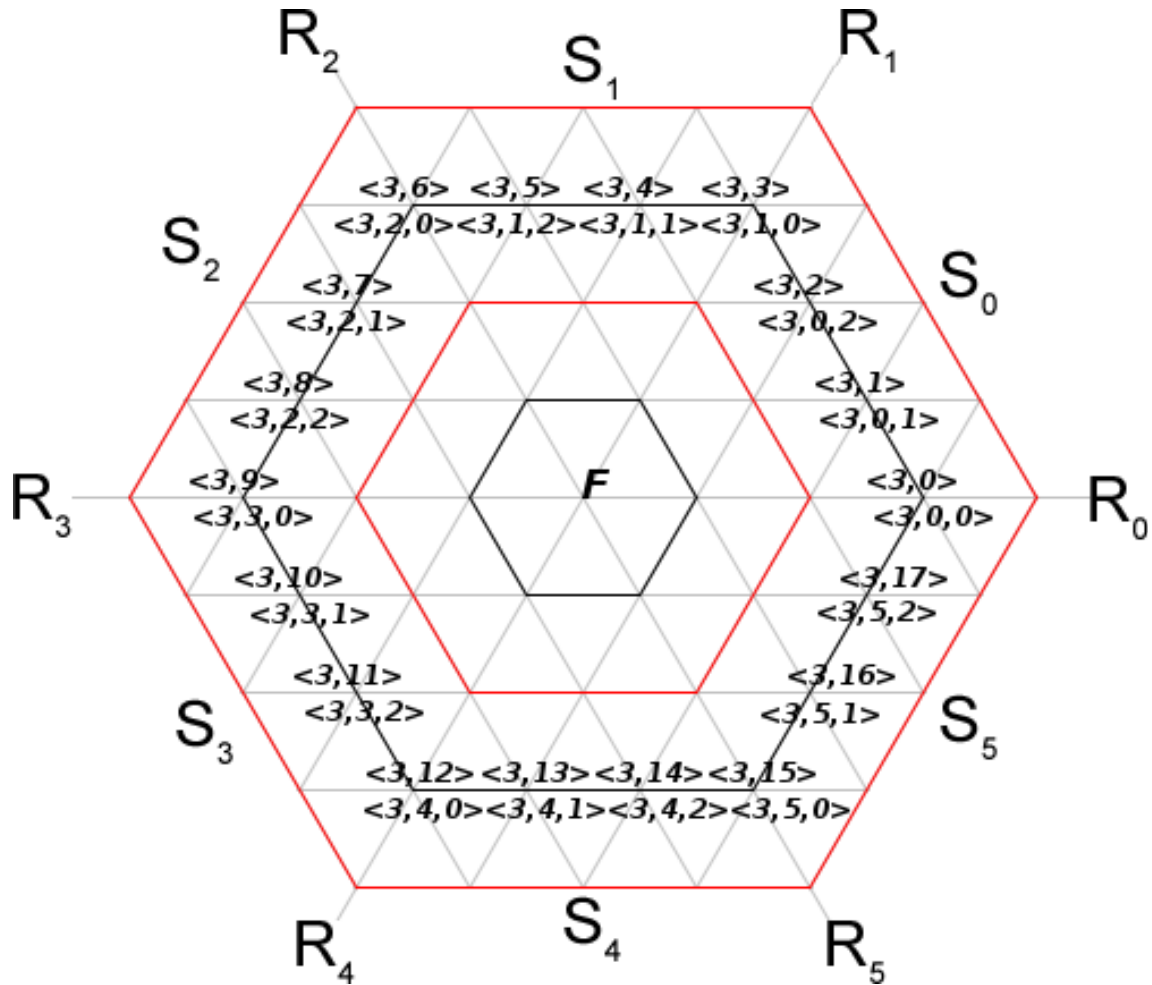
1.5. A dolgozat által elért eredmények

A dolgozatban megmutatom, hogy miként lehet a [4] cikkben bemutatott megoldást a háromszögrácson vándorló szenzorokra alkalmazni, miközben megtartjuk a [4] cikk által elért korlátokat. A diszkrétizált algoritmus validálására és teljesítményének mérése létrehozok egy a háromszögrácson alapuló algoritmusokat általánosan tesztelni képes keretrendszert is. A mérési eredményekkel pedig igazolom, hogy az új algoritmus a gyakorlatban is gyorsabb, mint az [1] és [2] cikkben leírt módszerek.

2. Az algoritmus

A dolgozat ezen része az új algoritmus részletes ismertetésről fog szólni. Előbb a szükséges definíciók és fogalmak magyarázata, majd maga az algoritmus és végül pedig a kapcsolódó matematikai állítások és bizonyítások adaptációi olvashatóak.

2.1. Navigáció a háromszögrácson



3. ábra: F ponthoz igazított háromszög rács

Mivel a háromszög oldalai a r_c és r_s által meghatározott fix értéket vehetnek fel az optimális fókuszált lefedettség érdekében, ezért ezentúl a háromszögek oldalának hossza legyen a . Az u és v szenzorok közti, a rács mentén haladó legrövidebb út hosszát jelöljük $SP(u, v)$ -vel. Az u és v szenzor ráctávolságát $d(u, v) = SP(u, v) \cdot a^{-1}$ összefüggéssel definiáljuk. Minden rácponttól pontosan $6 \cdot i$ rácpont van i ráctávolságban. Ezen pontok egy hatszög alakzatot feszítenek

ki ezért $H_{i,u}$ -val jelöljük őket. $H_{i,F}$ jelöléssel megegyezik H_i , az F pont kiemelt szerepe miatt.

F ponttól keleti irányban felvesszük az R_0 tengelyt és a rácshálózatot úgy illesztjük, hogy legyen olyan háromszög, melynek az egyik oldala párhuzamos ezen tengellyel. Bármely H_i hatszög csúcsaival meghatározhatjuk a többi tengelyt, amelyeket R_0 tengelytől óramutató járásával ellentétes irányba számozzunk. Két tengely közti területet szektornak nevezzük és S_i -vel jelöljük. A szektorok száma az óra mutató járásával megegyező irányba a határoló tengely számával egyezik meg.

A háromszög rácspontjainak azonosítására két -féle kézen fekvő jelölés is adott. Az első egy a polárkoordináta-hoz hasonló rendezett kettes $\langle a, b \rangle$, melynek jelentése, hogy

H_a hatszög R_0 tengelytől az óramutató irányával ellentétesen haladva a b -edik rácspont. A koordináták jelölésénél figyelembe vesszük, hogy $0 \leq b \leq (a \cdot 6) - 1$ illetve $1 \leq a$, kivéve $F = \langle 0, 0 \rangle$.

A másik lehetőség, hogy rendezett hármassokkal reprezentáljuk a rácspontokat. Ilyen eset $\langle a, b, c \rangle$ hármass, hogy H_a hatszög R_b tengelytől az óramutató irányával ellentétesen haladva a c -edik rácspont. A koordinátákra, ebben az esetben fennáll, hogy $1 \leq a$, $0 \leq b \leq 5$, $0 \leq c \leq a - 1$, kivéve $F = \langle 0, 0, 0 \rangle$. Ezen két reprezentáció könnyen megfeleltethető egymásnak $\langle a, b, c \rangle = \langle a, b \cdot a + c \rangle$, $\langle a, b \rangle = \langle a, b/a, a - b/a \rangle$.

Az algoritmus ismertetése során szükségünk lesz a rácspontok típusának fogalmára. Ez a tulajdonság azt írja le, hogy u pont H_i hatszög, oldalán (szektoron belül) vagy csúcsán (tengelyen) helyezkedik el. Koordinátákra lefordítva $\langle a, b, 0 \rangle$, vagy $\langle a, b \rangle$ ahol $a|b$ típusú pontok csúcspontok, a többi rácspont a hatszög oldalán helyezkedik el.

2.2. Az algoritmus ismertetése

A probléma ismertetése során már olvashattuk, hogy kezdetben a szenzorok különböző rácspontokban helyezkednek el, tehát az algoritmus során a szenzoroknak egyetlen feladatuk van, mégpedig hogy minél közelebb kerüljenek az F ponthoz. Az algoritmus körökre osztott és minden kör három fázist tartalmaz. Egy tetszőlegesen kiválasztott u szenzorra a következőképpen néz ki:

1. A szenzor felméri, hogy tőle két rácstávolságra milyen szenzorok vannak és azok mely rácspontokon helyezkednek el.
2. a) A szenzor megpróbál olyan biztonságos mozgást választani 1 rácstávolságban amellyel, csökkenti a F ponttól mért rácstávolságát. (Közelítő lépés)
b) Amennyiben a) lépés nem lehetséges akkor olyan rácspontot választ 1 rácstávolságban amellyel megtartja ezt a távolságot. (Keringő lépés)
3. u elmozog a kijelölt rácspontba
(Amennyiben az végzett a harmadik fázissal új kör kezdődik)

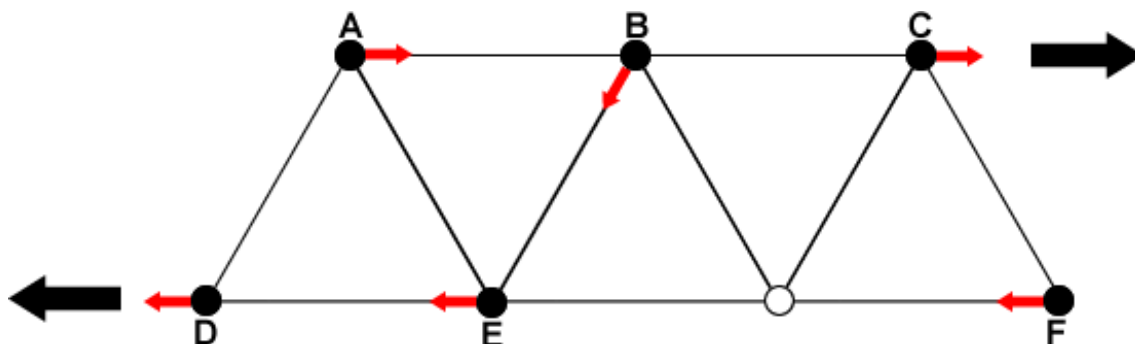
A következőkben kifejtem milyen esetekben végez haladó mozgást a szenzor illetve, hogy miként kering adott F pont körüli hatszögön.

2.2.1. Keringés

Mivel a közelítő lépéseknél sokszor fog szerepelni egy adott H_i forgásiránya ezért érthetőség kedvéért ezzel a mozgás típussal kezdjük az ismertetést. Keringésnek nevezzük tehát, ha a szenzor a kör elején és végén azonos H_i hatszögön helyezkedik el. Mint azt később látni fogjuk akkor tudunk közelítő lépést végre hajtani ha H_{i-1} tartalmaz üres rácspontot. Amennyiben H_i és H_{i-1} hatszögek forgásiránya különbözik kevesebb kör elegendő, hogy külső hatszögön lévő szenzor összetalálkozzon a belső hatszögön lévő üres rácsponttal.

Ezen okból kifolyólag, H_i hatszögön lévő szenzorok, amennyiben i páratlan az óramutató járásával megegyező irányba forognak, amennyiben i páros az óramutató járásával ellentétes irányba forognak.

2.2.2. Egyszerű közelítő lépés



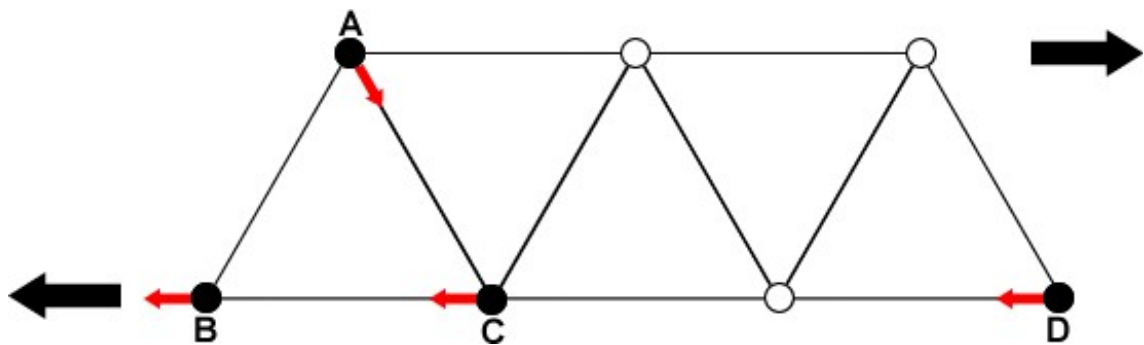
4. ábra: Egyszerű közelítő lépés

Adott u szenzor H_i hatszög oldalához tartozó pozíció. Amennyiben H_{i-1} hatszögön u forgásirányától számított első pozíció üres, akkor u szenzornak

lehetősége van elfoglalni ezen pozíciót. Tehát u szenzor következő pozíciója, amit a mozgás fázisa végén felvesz a H_{i-1} hatszögön u forgásirányától számított második pozíció lesz.

Az ábrán látható, hogy B szenzor, hogyan képes egy egyszerű lépéssel csökkenteni a F -től mért távolságát. Amennyiben lehetséges a szenzorok ezen mozgásformát követik.

2.2.3. Egyszerű közelítő lépés II. (nincs forgás irányú szomszéd)



5. ábra: Egyszerű közelítő lépés variánsa

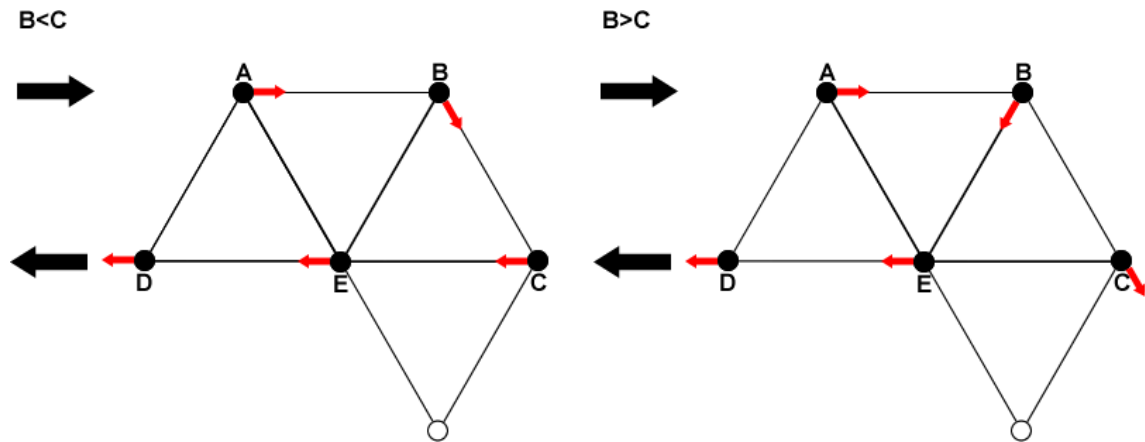
Adott u szenzor H_i hatszög oldalához tartozó pozíción. Amennyiben u szenzornak nincs lehetősége az előbb ismertetett egyszerű közelítő lépésre és nincs forgásirányú szomszédja, megvizsgálhatja, hogy a nem létező szomszédjának lehetősége lenne-e a közeledésre. Ha van üres pozíció a H_{i-1} hatszögön u forgásirányától számított első pozíción 2 rácstávolságban, akkor u szenzor a forgásirányától számított első pozíciót veszi fel 1 rácstávolságban a mozgás fázisa végére. (A nem létező szomszéd versenyhelyzetben automatikusan veszít.)

Az ábrán látható, hogy A szenzor nem képes egyszerű módon csökkenteni a távolságát ezért, az előbb leírt módon ellenőrzi a környezetét és elfoglalja nem létező szomszédjának helyét.

2.2.4. Lépés sarokról

Amennyiben u szenzor valamely H_i hatszög sarkán helyezkedik el csupán egyetlen lehetséges pontot lát a közeledésre méghozzá H_{i-1} egy sarokpontját. Mint az ábrán is látszik ekkor egy verseny helyzet alakulhat ki, hogy u vagy u forgásirányú szomszédja foglalja-e az üres pozíciót. Ebben a helyzetben a szenzorok

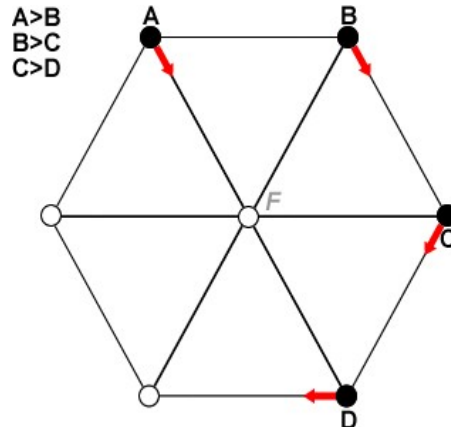
számára már amúgy is ismert adatból, a szenzorok sorszáma alapján rangsorolunk és utat engedünk a nagyobb sorszámmal rendelkező egyednek.



6. ábra: Lépés sarokról

Mint az ábrán is látszik ezt a szomszéd szenzornak is figyelembe kell vennie mikor felméri, hogy tud-e egyszerű lépéssel közeledni (C ugyan egyszerű lépéssel közeledne, de utat enged B szenzornak, így C mozgása keringés lesz nem pedig egyszerű közeledés).

2.2.5. Lépés sarokról a legbelső hatszögön



7. ábra: F pont elfoglalása

Mivel az első hatszögnek már nincsen oldalsó pontja, ezért a fenti szabály már nem alkalmazható. Ezért a legbelső rétegen a szenzorok a H_1 -en elhelyezkedő összes szenzorral összehasonlítják a sorszámaikat és akkor indulnak el F pontba ha az övék a legnagyobb sorszám.

2.3. A konvergencia elemzése

A dolgozat ezen részében [4] forrás modelljének konvergenciájára vonatkozó bizonyításait dolgozom át úgy, hogy illeszkedjen az általam ismert algoritmusához és jelöléseimhez. A bizonyításhoz szükséges volt [4] cikk által használt potenciál fogalmát POI-tól mért rácsávolságra cserélni. A bizonyítás lépéseinél továbbá figyelembe kell venni, hogy mivel már nem körpályán keringenek a robotok, ezért hatszögek sarkainál lévő versenyhelyzet feloldása már nem ugyanazokon a szabályokon alapul.

Lemma 1. *Bármely szenzor amely nem a H_0 rétegen tartózkodik, képes valamely lépés típussal csökkenteni F -től mért rácsávolságát, feltéve ha a körülötte elhelyezkedő szenzorok ezt lehetővé teszik. Amennyiben ez nem lehetséges keringő lépést végez, amely megtartja ezt a távolságot. Azaz a szenzorok a körök végén közelebb vagy ugyanolyan távolságra helyezkednek el mint körök elején.* $\square$

A lemma bizonyítása az algoritmus lépéseinek definícióiból triviálisan adódik. Ezután a következő lépés, hogy elkészítjük a felső korlátot konvergencia sebességére.

Tétel 1. *Amennyiben valamely H_i hatszögon van üres rácspont és létezik olyan u szenzor melyre igaz, hogy $d(u, F) > i$, akkor a szenzorok F -től mért rácsávolságainak összege, minimum 1-el csökkenni fog a következő $r_{om} \cdot 3 + 1$ körön belül, ahol r_{om} a célpont bekerítését követően legkülső még szenzort tartalmazó hatszög indexe.*

Bizonyítás:

Mint azt a fenti lemmában kimondtuk, a szenzorok F -től mért rácsávolsága nem nőhet. Amennyiben a szenzorunk nem találkozik egyetlen más szenzorral sem a bekerítés során, akkor a rácshálózaton haladva a legrövidebb úton elfoglalja H_0 hatszöget. Amennyiben egy körön belül nem csökken a szenzor F -től mért rácsávolsága, az azt jelenti, hogy vagy egy magasabb sorszámú egyed amely megelőzte őt egy versenyhelyzetben, vagy nem volt olyan üres rácspont az alacsonyabb indexű hatszögon amelyet elfoglalhatott volna.

Tegyük fel, hogy olyan helyzetbe kerültünk amikor, egyetlen szenzor sem képes csökkenteni rácsávolságát, illetve azt is, hogy H_i az első olyan hatszög amin van üres rácspont. Vegyük a legmagasabb sorszámú szenzort H_{i+1} hatszögről, amennyiben nincs ilyen végeztünk mivel az összes szenzor a belső hatszögeken kering.

Az előbb kiválasztott szenzort nevezzük u -nak. Tudjuk, hogy u és az üres rácspont ellentétes irányban keringenek, ezért legkésőbb $i \cdot 3 + 1$ kör múlva u , elfoglalja az üres rácspontot, amennyiben nem előzi meg egyetlen másik szenzor sem H_{i+1} -es hatszögről. $\square$

Tétel 2. $O(D)$ idő elteltével az F körüli hatszögek feltöltöttsége teljes lesz, azaz $\forall u \in V, d(u, F) \leq r_{om}$, ahol V a szenzorok halmaza, D pedig a szenzorok F -től mért rács távolsága a probléma kezdeti állapotában.

Bizonyítás:

Legyen $i \geq 0$ és $j \geq 1$, $t_{i,j}$ azaz időpont amikor az összes $H_l, l < i$ hatszögön nincs már üres rácspont és a H_i hatszögön a szenzorok száma $j-1$ -ről j -re emelkedik. A $v_{i,j}$ szenzor a H_i hatszögrácspontjaihoz $t_{i,j}$ időpillanatban csatlakozott és utána ott is maradt, mivel a kisebb indexű hatszögek már megteltek. Jelöljük m_i -vel a szenzorok számát H_i hatszögön ($i \geq 0$), amikor $H_l, l < i$ hatszögön nincs már üres rácspont. Azon szenzorok halmazát amik $t_{i,j}$ időpontra elfoglalták végső hatszögeket $V_{i,j}$ -vel jelöljük. Egyszerűsítsük a leírást $t_{i+1,0} := t_{i,m_i}, i \geq 0$ és a $t_{0,0}$ tekintjük a start időpontjának. Definiáljuk a $dt_{i,j} := t_{i,j} - t_{i,j-1}$, $i \geq 0, 1 \leq j \leq m_i$. Legyen $u_{i,j} \in V \setminus V_{i,j}$ az a szenzor amely a legközelebb helyezkedik el H_i hatszöghöz és legmagasabb sorszámmal rendelkezik $t_{i,j-1}$ időpontban $V \setminus V_{i,j}$ halmaz elemei közül. Ekkor $u_{i,j}$ szenzor minden lépésben képes csökkenteni a F -től mért rács távolságát egészen addig amíg el nem éri a H_{i+1} hatszöget. Ezután $(i+1) \cdot 3 + 1 = O(i)$ időn belül elfoglalhatja az H_i hatszögön lévő üres rácspontot, amennyiben ezt más szenzor nem tette meg előtte. Megjegyezzük, hogy amennyiben $u_{i,j}$ nem tud belépni H_i hatszögre ezen időintervallum alatt, akkor elérte a végső pályáját amelyen a továbbiakban keringeni fog. Így megfigyelhetjük, hogy $dt_{i,j} \leq d_{i,j-1}(u_{i,j}, F) + O(i)$, ahol $d_{i,j-1}(u_{i,j}, F)$ az $u_{i,j}$ rács távolsága F -től $t_{i,j-1}$ időpillanatban, amiről tudjuk, hogy nem nagyobb mint a probléma kezdeti állapotában volt azaz $d_{0,0}(u_{i,j}, F)$, ezért $dt_{i,j} \leq c \cdot d_{0,0}(u_{i,j}, F)$, valamely $c > 1$ konstansra. A fentiekből tehát következik, hogy az t idő, ami alatt a szenzorok elfoglalják a végső hatszögeket, a következő módon fejezhető ki: $t = \sum_i \sum_j dt_{i,j} \leq \sum_{u \in V} c \cdot d(u, F) = O(D)$ $\square$

Mint az a bizonyításból látszik a [4] cikkben felállított korlát továbbra is érvényben marad, a hatszögeken futó algoritmus esetében is. A továbbiakban arra fektetjük a hangsúlyt, hogy megmutassuk, hogy ez az algoritmus nem csak elméleti szinten produkál jobb eredményeket mint [2] cikkben bemutatott, hanem gyakorlatba is.

3. Szimulációs környezet

Az algoritmus validálására és teljesítményének mérésére szükség volt egy olyan környezetre amiben egyszerűen és gyorsan tudom implementálni a szenzorok működését, és a szimulációk futása után az adatok könnyen feldolgozható formátumba állnak rendelkezésre. Több szimulációs környezet megvizsgálása után úgy döntöttem, hogy egy saját keretrendszer alakítok ki erre célra.

A fejlesztés során megvalósítandó célkitűzéseim a következők voltak:

- Új algoritmusok tervezésénél és fejlesztésénél támogassa a programozót
- A rendszer egyszerű interfészekkel rendelkezzen
- Kimenő és bemenő adatok szabványos XML formátumúak legyenek
- Grafikus felülettel mutassa be a szenzorok mozgását

A fenti célokból is látszik, hogy nem az algoritmust célirányosan tesztelő programot írtam, hanem egy a háromszögácson alapuló algoritmusokat általánosan tesztelni képes keretrendszert. Mivel dolgozat elsősorban nem erről a rendszerről szól ezért csak egy rövid áttekintést kívánok adni az elkészült program főbb funkciójáról és azok használatáról.

3.1. Szenzor implementáció

A környezet teljes körű kihasználásához, az új szenzor típusnak két feltételnek kell eleget tennie. Az első, hogy az új osztálynak a `SensorGraphic` osztályt kell kiterjesztenie a másik, hogy megfelelően implementáljuk a `SensorControl` interfészt.

3.1.1. `SensorGraphic` osztály

A grafikus felület által használt függvényeket tartalmazó osztály. Használata akkor szükséges ha igénybe kívánjuk venni a program által nyújtott grafikus megjelenítést, és az ahhoz tartozó funkciókat. A programozónak egy új szenzortípus létrehozásakor ezt az osztályt kell kiterjesztenie.

3.1.2. `SensorControl` interfész

A futtató környezet ezen az interfészen keresztül kommunikál a szimulációs felületre felhelyezett szenzorokkal. Ezen műveletek definiálásával írjuk le, hogy a szenzorunk a önelrendezés során, milyen algoritmust követnek.

firstHop művelet

A szimuláció első köre helyett a szenzorok ezen művelete kerül meghívásra.

beforeStep művelet

A környezet minden kör elején ezzel a függvénnyel inicializálja szenzor mozgásához szükséges adattagokat a *calculateHop* meghívása előtt. Például a szenzor felderítheti, hogy milyen szenzorok vannak tőle 2-hop távolságban.

(A függvény első sorban kód strukturáltságát hívatott növelni.)

calculateHop művelet

Azután hogy *beforeStep* művelet előkészítette a szükséges információkat számunkra, meghozzuk az algoritmikus döntést, arról hogy mi legyen az adott körben a szenzorunk mozgásának célpontja.

hop művelet

Miután minden szenzor végzett a célpont kiválasztásával, a futtató környezet áthelyezi őket, a régi pozíciójukból az újba. Ezen művelet felüldefiniálása nem szükséges, mivel az ősoosztályban megvalósított művelet.

afterStep művelet

Miután a szenzor megérkezett a kijelölt pozícióba, környezet meghívja a *afterStep* metódust.

(A függvény első sorban kód strukturáltságát hívatott növelni.)

3.2. Környezet által biztosított eszközök

A program célkitűzései között szerepelt, hogy az elkészülő keretrendszer támogassa a programozót, mind a fejlesztés mind szimuláció lépések során. A szenzor műveletek illetve a grafikus felület az implementációs részt segíti, a generátorok és a xml feldolgozás mindkét rész alatt használható. A kötegelt futtatást lehetővé tevő *batchRun* függvény pedig egyértelműen a szimulációs lépéshez tartozik.

3.2.1. Szenzor műveletek

A *SensorBase* osztály rendelkezik a *beforeStep* lépés során általában használt műveletekkel, mint a szomszédos rácsponthoz feltérképezése, vagy a POI irányának meghatározása, amennyiben pedig globális információkra van szükségünk az adott rácspontról, úgy *HexaFieldUtil* osztály szolgál ki minket.

Ezekkel függvényekkel gyorsan és könnyen implementálhatjuk a megtervezett algoritmusunkat.

3.2.2. Megjelenítés

Fejlesztés során a legnagyobb segítséget a grafikus megjelenítés, és az onnan elérhető funkciók adják. A fejlesztő láthatja, hogy miként mozognak a szenzorok, megjelenítheti a szenzorok által eddig megtett utat. Amennyiben pedig hiba van, képes megállítani és újra játszani az adott szituációt. Ha végzett egy szituáció tesztelésével nincs más teendője mint betölteni egy másik xml fájlt.

3.2.3. Szimulációs xml bemenet és annak feldolgozása

```
<?xml version="1.0"?>
<field side="50" panelsize="700">
  <sensor type="FS" x="350" y="344" id="S01"/>
  <sensor type="FS" hexa="1" position="4" id="S02"/>
  <sensor type="FS" hexa="4" sector="2" position="3" id="S03"/>
</field>
```

Azt hogy a szimuláció kezdetén milyen pozíciókon helyezkednek el a szenzorok egy xml fájl segítségével írjuk le. Ez azért előnyös mert könnyen olvashatóak és szerkeszthetőek, ami tesztesetek megírásánál és módosításánál is hasznos. A szenzorok megadására háromféle mód áll rendelkezésünkre. Az első direkt koordináták segítségével (S01 azonosítójú szenzor), a másik kettő az algoritmus leírásánál bevezetett két (S02) illetve három (S03) koordinátás alak. Az utolsó *type* attribútumra az xml feldolgozását végző *HexaFieldReader* osztálynak van szüksége, ami ezen érték alapján példányosítja majd a megfelelő szenzorokat. (A *field* attribútumai a grafikus megjelenést befolyásolják)

3.2.4. Generátorok

Mivel az xml-ek manuális előállításuk sok esetben a teszt esetekhez sem elegendő, ezért a keretrendszer erre is ad egy szolgáltatást. A *generator* csomagban *HexgonGenerator*-nak elnevezett osztályt. Ez az osztály egy a szektorokra lebontott input fájlokat generál, amihez bemenő adatként szüksége van arra, hogy megmondjuk mely szektorokba helyezhet le szenzort, mekkora lehet a lehelyezett szenzorok rács távolsága a POI-tól,

hány darab és milyen típusú szenzort szeretnénk felhelyezni. Az ilyen input fájlok ideálisak a tesztesetekhez.

A SquareGenerator osztály már inkább a mérésekhez használható, itt az aktív szektorok kivételével ugyanazok a bemenő adatok, mint az előző esetben. A generátor által előállított fájlokban a szenzorok a POI körüli négyzet alakzaton belül helyezkedhetnek el. A TDK dolgozat méréseihez használt adatok ezen generátor segítségével készültek.

3.2.5. Szimulációk futtatása

Amennyiben a programot nem grafikus felülettel futtatjuk, akkor a szimuláció leállításának feltétele, hogy a program adott szenzorszám mellett elérje az optimális fókuszált lefedettséget a POI körül. Amennyiben olyan algoritmust terveztünk amely ezt nem éri el, azt nem lehet ilyen módon futtatni.

Miután a szimuláció végzett az input fájl mellett megjelenik egy a szimuláció adatait tartalmazó xml fájl.

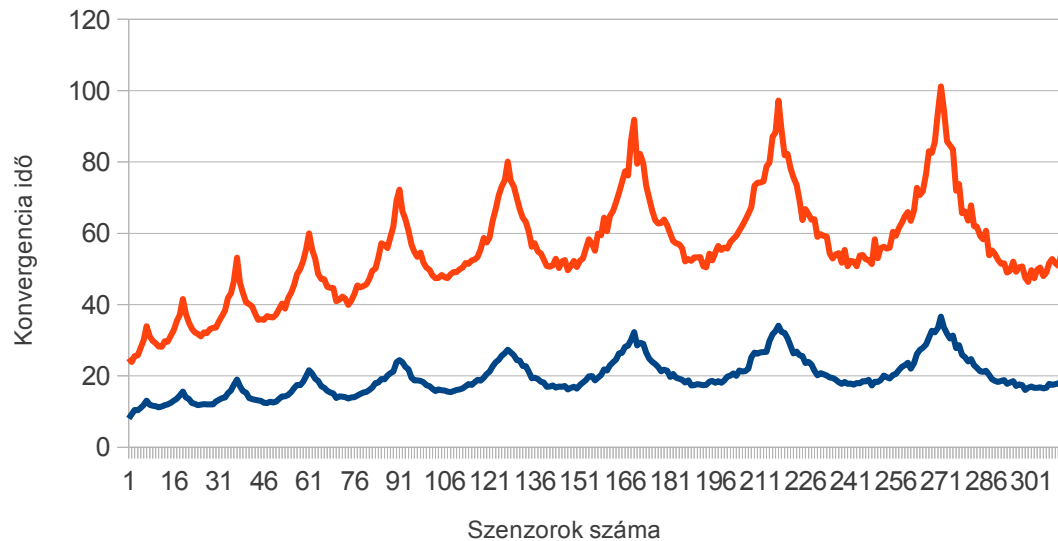
```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<result>
  <convergence round="9" />
  <sensors>
    <sensor convergenceTime="8" id="S0" movementVector="aaaaaaaaa0"/>
    <sensor convergenceTime="9" id="S1" movementVector="aaaaaaaaa"/>
  </sensors>
</result>
```

A fenti minta xml-ben láthatjuk melyik szenzor mennyi idő alatt érte el végső pályáját (convergenceTime), a movementVector értékei pedig az mutatják meg, hogy az egyes körökben a szenzor közeledő (a – advance), keringő (o – orbit), távolodó (r – retreat) lépést végzett. A könnyű kezelhetőség kedvéért a leghosszabb konvergencia időt kiemelve is láthatjuk, mint a convergence tag round attribútumának értéke.

4. Eredmények

A dolgozat ezen részébe bemutatjuk az adaptált algoritmus teljesítményének mérésére szolgáló szimulációk eredményeit, melyeket a fent ismertetett szimulációs eszköz segítségével készítettem el. A szimulációban használt szenzorok 100 egység hosszúságú kommunikációs sugárral rendelkeznek, amiből adódik hogy $r_s = (2\sqrt{3})^{-1} \cdot 100 \approx 29$ egység hosszú.

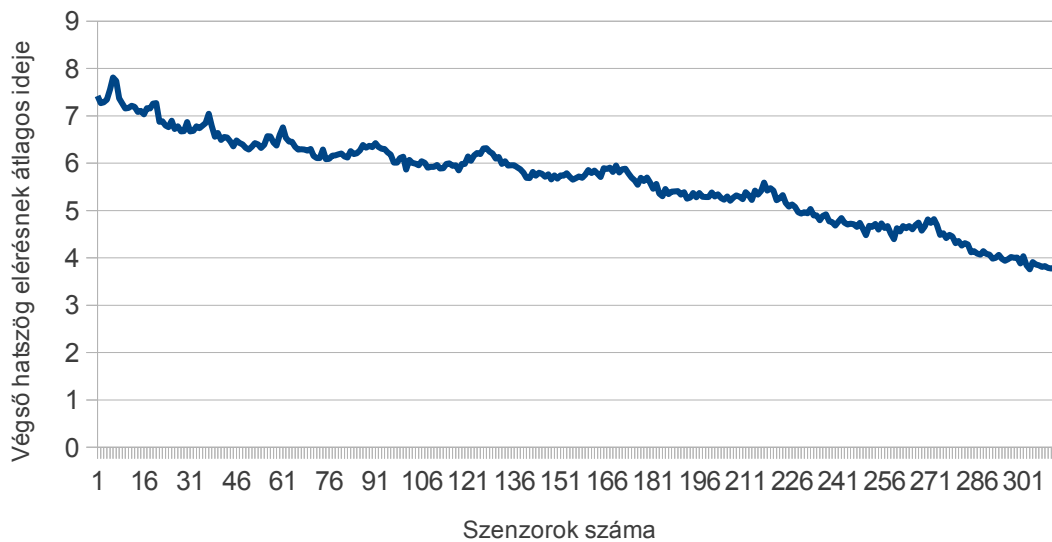
A [2] cikk szerzői a szimulációs adataik elemzése során három szempontot vizsgáltak, a konvergencia idejét, a szenzor ütközések számát, illetve a szenzorok kommunikációra fordított energia felhasználását. Mivel a szenzorok kezdetben különböző rácsponatokon helyezkednek el, és ilyen kiindulási körülmények között mindkét megoldás garantálja, hogy nem generál új ütközéseket, ezért e tekintetben teljesítményük megegyezik. A szenzorok közötti kommunikációra fordított energia felhasználás az új algoritmusnál a konvergencia idővel egyenes arányban áll, ezért ezen tulajdonság vizsgálatára nem szükséges külön méréseket végezni.



8. ábra: Szenzor szám növelése fix terület mellett (kék adatsor az új algoritmus, narancs [2]-es cikk GRG/CV algoritmus)

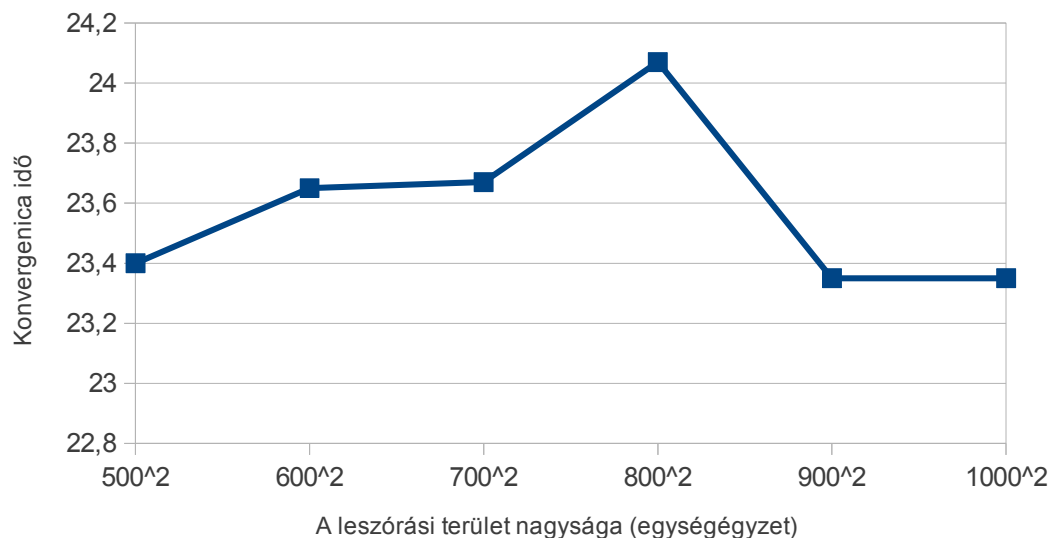
A konvergencia sebességének mérésére két különböző foratókönyvet használtam. A szimulációk első csoportja egy 1000^2 egységnégyzet területű POI középpontú négyzet alakzatra szórta szét a szenzorokat. Ezen szimulációk paramétereit a szenzor számot egyenletesen növeltem a 1-től egészen a H_{10} hatszöget teljesen feltöltő 331-

-es darab számig. A 8. ábrán látható grafikonon ötven szimulációs sorozat átlag teljesítményét mutatja. A grafikonon kiemelkedő csúcsokat az utolsó szenzor helyre kerülése okozza mivel, minél több szenzor helyezkedik el a legkülső hatszögön annál nagyobb a valószínűsége, hogy az utolsó a hatszögre belépő szenzornak, meg kell tenni egy fél fordulatot, ahhoz hogy üres rácspontot találjon.



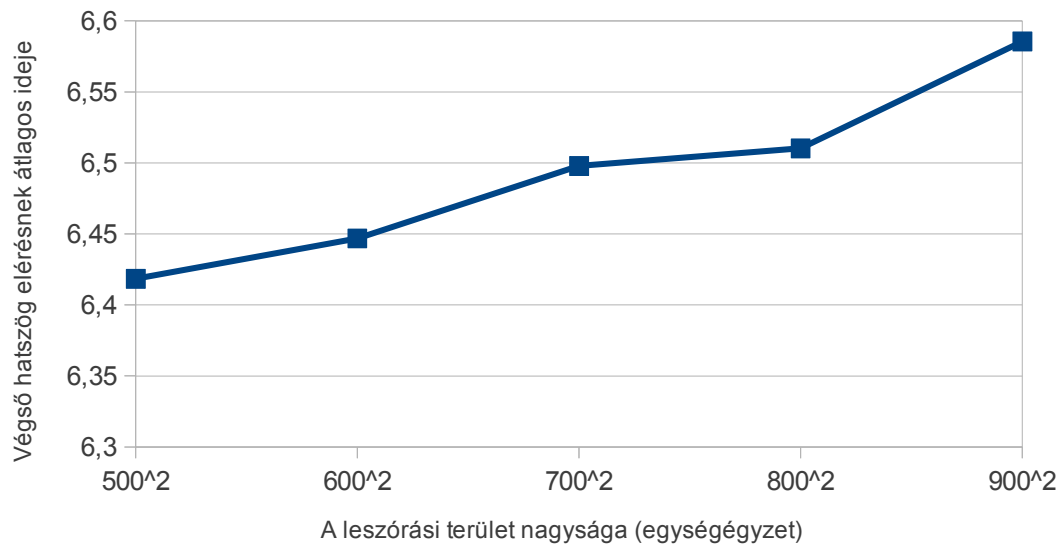
9. ábra: Átlagos konvergencia idő, a szenzor szám növelése és fix terület mellett

Ha ugyanezen adatokból, kiindulva a szenzorok átlagos konvergencia idejét vesszük (lásd a 9. ábrán), akkor már jól látható, hogy szenzor szám növelése mellett gyorsabban alakul ki optimális lefedettség.



10. ábra: Konvergencia idő állandó szenzorszám és a növekvő terület mellett

A második csoportban állandó számú a H_5 hatszöget teljesen felöltő 91 szenzorral dolgoztam, és a szimuláció paramétere a befoglaló négyzet alakzat területe változott 500^2 egységnégyzetről egészen 1000^2 -ig. A 10. ábra grafikonján látható pontok száz mérési sorozat átlagaként születtek. A kezdeti terület a négyszeresére nőtt, a konvergencia idő viszont szignifikánsan nem tér el, tehát a terület növelése mellett is a legkülső hatszög mérete befolyásolja fő mértékben a konvergencia idejét.



11. ábra: Átlagos konvergencia idő állandó szenzorszám és a növekvő terület mellett

A 11. ábrán látható, hogy ugyan a legkülső hatszög kerülete főtényező, de a ledobási terület növelése az átlagos konvergencia időt növeli.

[2] cikk mérési grafikonjaival összevetve a saját eredményeinket, láthatjuk hogy az elkészített algoritmus eredménye, meghaladja a GRG/CV néven közölt algoritmus eredményeit, figyelembe véve azt, hogy [2] cikkben leszórt szenzorok POI-tól mért átlagos távolsága kisebb, mint az adaptált algoritmus méréseinél használt adatokban.

A GRG/CV algoritmussal kapcsolatos saját mérési adataimat 8. ábrára vittem fel, mert így ezen a grafikonon könnyen látható a teljesítménybeli különbség. A többi ábrán azért csak egyetlen adatsor szerepel, mivel ezen különbség miatt láthatatlanná válnának a grafikonok leírásánál olvasható megfigyelések.

5. További tervek

Az adaptált algoritmus kiindulási feltétele az, hogy a szenzorok kezdetben különböző rácpontokon helyezkednek el, egy valós helyzetben nem várható el. (például repülőről leszárt szenzorok esetén). Így a szenzorok algoritmusát bővíteni kell ezen kezdőfeltétel megteremtésének képességével, úgy hogy az algoritmus hatékonysága ne csökkenjen. Mivel sok helyzetben a szenzorok folyamatos mozgása nem kívánatos (katonai felhasználás), vagy nem szükséges (vegyszeri üzem), ezért fontos, hogy az algoritmus terminálásának módszerét is megvalósítsuk, úgy hogy az algoritmikus korlátok ne sérüljenek.

A dolgozat során az algoritmus mellett elkészült egy szimulációs környezet is, amely nem az algoritmust célirányosan tesztelő program, hanem egy a háromszögrácson alapuló algoritmusokat általánosan tesztelni képes keretrendszer. Ezen eszköz által megvalósított funkciók bővítése, esetleg a háromszög rács elhagyásával egy még általánosabb rendszert megvalósítása is célkitűzés.

6. Összefoglalás

A dolgozatomban bemutatam egy új hatékony lokalizált algoritmust, amely négy egyszerű szabály mellett képes biztosítani az optimális lefedettséget adott POI körül. A dolgozat során továbbá beláttam, hogy a megoldás alapjául szolgáló [4] cikkben felállított korlátok érvényesek maradnak, amennyiben a potenciál fogalmát a megváltozott diszkrét környezethez illeszkedő adatokra cseréljük és figyelembe vesszük a rácshálózat használata miatt kialakuló speciális helyzeteket.

Az algoritmus ellenőrzésére létrehoztam egy keretrendszert, amely funkcióival túlmutat egy egyszerű, a mérésekhez használt tesztalkalmazáson. A program képes könnyebbé és hatékonyabbá tenni hasonló szenzor hálózatok irányítását megvalósító algoritmusok fejlesztését, tesztelését és a hatékonyságuk mérését.

A dolgozat végen, a diszkrét algoritmus szimulációs adatait felhasználva, megmutattam, hogy a [2] cikk szerzői által, GRG/CV néven közölt megoldásnál, az új algoritmus nem csak elméleti szinten képes jobb eredményeket produkálni, hanem a gyakorlatban is.

Irodalom jegyzék:

1. Xu Li, Hannes Frey, Nicola Santoro, Ivan Stojmenovic: Localized sensor self-deployment with coverage guarantee. *Mobile Computing and Communications Review* 12(2): 50-52 (2008)
2. Xu Li, Hannes Frey, Nicola Santoro, Ivan Stojmenovic: Strictly Localized Sensor Self-Deployment for Optimal Focused Coverage. *IEEE Trans. Mob. Comput.* 10(11): 1520-1533 (2011)
3. László Blázovics, Tamás Lukovszki: Surrounding robots -- A localized solution for the intruder problem. Accepted: *3rd IEEE International Conference on Cognitive Infocommunications (CogInfoCom 2012)*, 2012.
4. László Blázovics, Tamás Lukovszki, Bertalan Forstner: Target surrounding solution for swarm robots. In: *Proc. 18th EUNICE Conference on Information and Communications Technologies*, LNCS, Springer, 2012.
5. M. Ma and Y. Yang, "Adaptive Triangular Deployment Algorithm for Unattended Mobile Sensor Networks," *IEEE Trans. Computers*, vol. 56, no. 7, pp. 946-847, July 2007.
6. *Wireless Sensor and Actuator Networks: Algorithms and Protocols for Scalable Coordination and Data Communication*, Wiley, 2010.
7. M. Garetto, M. Gribaudo, C.-F. Chiasserini, and E. Leonardi, "A Distributed Sensor Relocation Scheme for Environmental Control," *Proc. IEEE Int'l Conf. Mobile Adhoc and Sensor Systems*, 2007.
8. A. Howard, M.J. Mataric, and G.S. Sukhatme, "Mobile Sensor Network Deployment Using Potential Fields: A Distributed, Scalable Solution to the Area Coverage Problem," *Proc. Distributed Autonomous Robotic Systems (DARS '02)*, 2002.
9. Mousavi H, Nayyeri A, Yazdani N, Lucas C. "Energy conserving movement-assisted deployment of Ad hoc sensor networks". *IEEE Commun Lett* 2006;10(4):269– 271.
10. N. Heo and P.K. Varshney, "Energy-Efficient Deployment of Intelligent Mobile Sensor Networks," *IEEE Trans. Systems, Man, and Cybernetics Part A: Systems and Humans*, vol. 35, no. 1, pp. 78- 92, Jan. 2005
11. G. Wang, G. Cao, and T.L. Porta, "Movement-Assisted Sensor Deployment," *IEEE Trans. Mobile Computing*, vol. 5, no. 6, pp. 640- 652, June 2006.
12. Yang S, Li M, Wu J. "Scan-based movement-assisted sensor deployment methods in wireless sensor networks". *IEEE Trans Parallel Distrib Syst* 2007;18(8):1108– 1121.

13. N. Bartolini, T. Calamoneri, E.G. Fusco, A. Massini, and S. Silvestri, "Snap & Spread: A Self-Deployment Algorithm for Mobile Sensor Networks," Proc. IEEE Fourth Int'l Conf. Distributed Computing in Sensor Systems (DCOSS '08), pp. 451-456, 2008.