Debreceni Egyetem Informatika Kar

Robotprogramozás Java nyelven

Témavezető:
Bátfai Norbert

Készítette:
Hudák László
Programtervező matematikus

Debrecen
2010

Tartalomjegyzék

1. Bevezetés

Napjainkban egyre nagyobb tért hódítanak a robotok. A számítógépek fejlődésével a robotok is egyre fejlettebbek és elterjedtebbek lettek. Míg kezdetekben csak az iparban használták őket, úgy mára már háztartások mindennapjainak szerves részei. A magas szintű nyelvek elterjedésével programozásuk is egyre egyszerűbbé vált. Manapság már kereskedelmi forgalomban kaphatók olyan robotok is mely segítségével a gyerekek is könnyedén, játszva tanulhatják meg a programozás alapjait. Ilyen például az National Instruments és a Lego által gyártott Lego Mindstorms készlet. Ennek a nagy előnye, hogy a Lego készlet gazdagsága miatt csak az ember fantáziája szab határt a megépíthető robotok változatosságának. A szokásos Lego készletektől annyiban tér el, hogy ehhez járnak programozható alkatrészek is, amiket beszerelhetünk építményünkbe, melyekkel robotjaink sokféle feladatot láthatnak el.

A Lego Mindstorms eredetileg egy gyerekek számára készült játéknak szánták. De meglepő módon az első verzió felhasználóknak csaknem 70 százaléka a felnőtt korosztályból való volt. A készlet a hackerek világára is hatással volt. Ők kezdték el a rendszert feltörni és lecserélni a firmware-t, hogy magas szintű nyelven is lehessen programozni a téglát. A tervezők, ahelyett, hogy jogi útra terelték volna az ügyet, bevonták az ügyeskedőket a fejlesztésbe. Többek közt ez is arra vezetett, hogy a robotokat mára mindenki számára szimpatikus nyelven programozhatja.

A Mindstorms fejlődésével egyre nagyobb népszerűségnek örvendett, különösképp mivel nem csak játékra használták. Mérnökök is alkalmazták, hogy mielőtt az éles munkájukba belekezdenének, felépítik a Mindstroms készlettel egy egyszerűsített prototípusát, mintegy vázlatát a megépítendő szerkezetnek. Így egyrészt a feladatukat is könnyebben átláthatják, másrészt a megrendelőnek is be tudnak mutatni egy alapkoncepciót.

Az oktatásban is rendkívül hasznosnak bizonyult a Lego robot. A Mindstorms segítségével a programozás tanulása sokkal jobban leköti a tanulók figyelmét. Mivel a programozása meglehetősen egyszerű, akár rögtön a egy „HelloWorld” program után meg lehet próbálni egy egyszerűbb robot vezérlőszoftverének megírásával. Azonban ellentétben egy „két bekért szám összeadása” vagy egy „első száz prímszám” jellegű programmal itt rögtön valami látványosat és kézzelfoghatót tudunk alkotni.

A dolgozat központi témája a Jávácska kupa nevezetű autóverseny [3], melyen az induló csapatok által a Lego Mindstorms készlet felhasználásával épített robotautók versenyeznek. A Lego robotokra szoftver írás módszereit ezen a versenyen és az autók programozásán

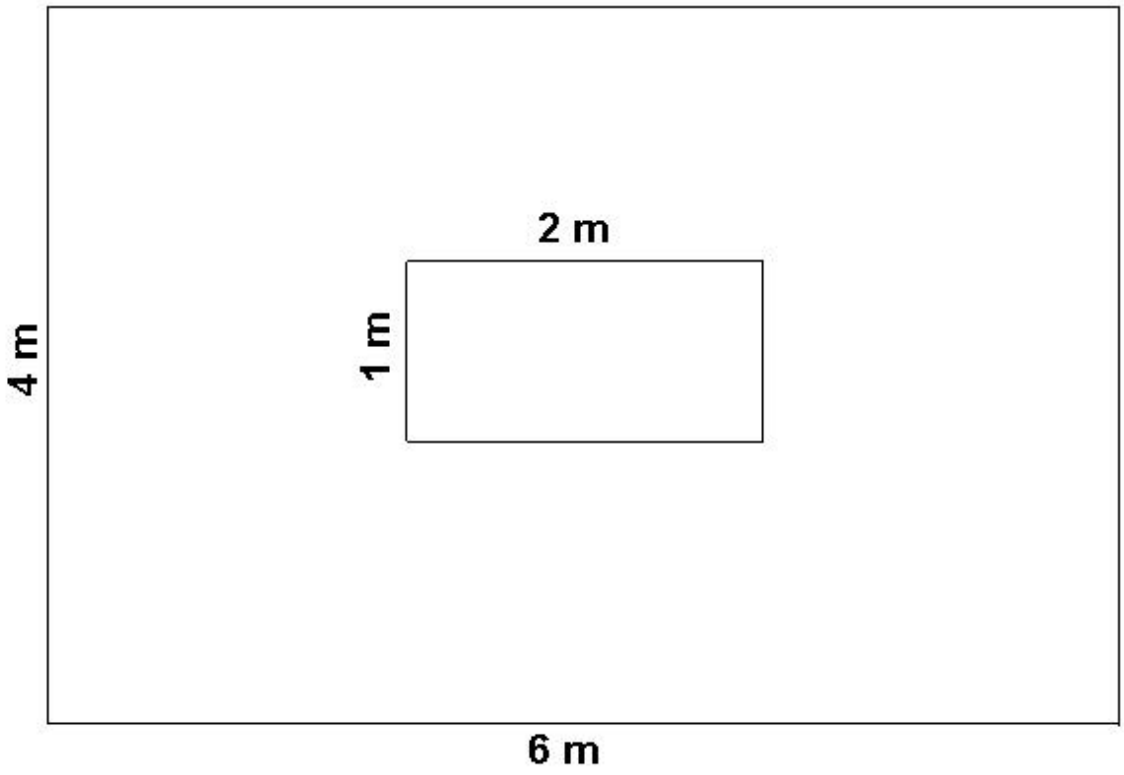
keresztül fogom bemutatni. A dolgozatban a szoftverek Java nyelven írásával fogok foglalkozni, a leJOS API segítségével. Először kitérek a motorok és az érzékelők kezelésére (különös tekintettel a távolságszenzorra). Ezután szó esik a viselkedésprogramozásról a Behavior API használatával. Bemutatom, hogy csapatunk milyen viselkedés alapú szoftvert írt a Jávácska kupás robotra. A Kiterjesztett Jávácska Kupa kapcsán leírom, hogy lehet a leJOS segítségével megvalósítani a robot Bluetooth kommunikációját. Bemutatok egy alkalmazást, amivel mobiltelefonon keresztül táv irányíthatjuk az autót, valamint egy programot, amivel, egy kamerával rendelkező mobiltelefont a robot elejére rögzítve, az a kamerakép alapján fog tájékozódni. Az utóbbi a Nokia Calling All Innovators versenyen különdíjat nyert [13].

2. Jávácska kupa

Mikor először kellett a Lego Mindstorms-szal foglalkoznom elsősorban azt kaptam feladatul, hogy találjam ki, mire lehet felhasználni a készletet. Ami eszembe jutott, hogy lehetne több ember bevonásával robotautókat építeni és azokra olyan szoftvert írni, hogy azok önállóan emberi beavatkozás nélkül tudjanak versenyezni egymással. Ebből az elgondolásból született meg a Jávácska kupa ötlete is [3].

Ennek lényege, hogy a Jávácska kupán induló versenyistállók építenek egy-egy autót, írnak rá vezérlőszoftvert és azok egy szabványos pályán fognak versenyezni. Két fontos szabály van. Az egyik, hogy csak a Lego Mindstroms NXT készlet elemeit lehet felhasználni a robot megalkotása során. A másik, hogy a vezérlőprogramot Java nyelven kell megírni a LeJOS API felhasználásával.

A pálya kialakítása során fontos szempont volt, hogy szabványos legyen és bárhol könnyen fel lehessen állítani. Jelenlegi állás szerint a pálya téglalap alakú, mindössze négy 90 fokos balkanyarral. A külső határa 4x6 négyzetméteres területet ölel körül, a belső kör 1x2 négyzetméter területű, ahogy az ábrán látható.



A pályát egy kb. 10 centiméter vastag vászon szalag határolja. A szalagon kétméterenként hurkok vannak kialakítva, amibe széklábakat lehet befűzni. Ily módon székek segítségével lehet a pályát kifeszíteni. A későbbi tervekben szerepel némiképp komolyabb pálya kialakítása, több kanyarral. Egyelőre azonban a pálya ilyen egyszerű formában áll fel. A verseny is még csak annyiban tudott megvalósulni, hogy az autók nem egyszerre mennek a pályán, hanem külön-külön időre teszik meg az előírt számú köröket. Ezeknek fő oka, hogy az NXT rendszer korántsem tökéletes és vannak benne pontatlanságok. Ezeknek a kisebb hibáknak a kiküszöbölésére vagy megkerülésére még folynak a kutatások. A Jávácska kupa alapvető lényege azonban, hogy a hallgatók a programozás és azon belül is az objektumorientált programozás alapjait könnyedén, de mégis megfelelő kihívás mellett tudják elsajátítani. Egy kisebb alapprogram megírása nem komplikáltabb, mint a manapság oktatott példaprogramoké, de az eredmény lényegesen látványosabb, így sokkal jobban felkelti az oktatottak érdeklődését a programozás iránt. A Debreceni Egyetem Informatika Karán a mérnök informatikus hallgatók Magas szintű programozási nyelvek 2 tantárgyból választhatják feladatul egy istálló indítását. A választható feladatok között a Jávácska Kupának több verzióját is meg lehet találni. Ezek a Kiterjesztett vagy Központosított Jávácska Kupa [4].

3. Lego Mindstorms NXT

A Lego Mindstorms nagyszerűségét az adja, hogy ez egyszerre lehet játék gyerekeknek, tervezési segédeszköz mérnököknek, valamint oktatási eszköz a programozással ismerkedő tanulóknak. A Lego Mindstroms vezérlőtéglára programot írva és ehhez a készletben található motorokat és szenzorokat csatlakoztatva, a Lego építési szabadságát felhasználva a legváltozatosabb alakú és működésű robotokat alkothatunk.

Az előfutárát, az úgy nevezett Programozható Téglát 1987-ben kezdték el fejleszteni az MIT Media Laboratóriumában a LEGO csoport támogatásával. Az évek során sok különböző verziója jelent meg. Ezt követte a 1998-ban a Robotics Invention System (Robotikai Feltaláló Rendszer szó szerinti fordításban, továbbiakban RIS). Ennek a lelke a még igen szerény képességekkel bíró sárga RCX tégla. Ebben egy 8 bites 16 MHz-es processzor van beépítve, ami mindössze 32 Kb memóriát használ. Ennek ellenére már ennek is meglepően nagy sikere volt. Az iskolákban, egyetemeken ez is széles körben elterjedt oktatási segédeszközként.

A folytatás megjelenésére még 8 évet kellett várni, de 2006 augusztus 2.-án piacra dobták az Lego Mindstroms NXT készlet első verzióját. Ez már kapott egy jóval elegánsabb külsőt, mint az RCX tégla, és ami még fontosabb többszörösére növelték a számítási kapacitását. Másik fő előnye az RIS-szel szemben, hogy rendelkezik Bluetooth adóvevővel, ami segítségével csatlakozhat személyi számítógépekhez, PDA-khoz, mobiltelefonokhoz, GPS készülékekhez. Ezek a lehetőségek széles skáláját nyújtják a robotprogramozók számára.

Az NXT készlet rengeteg Lego építő elem mellett tartalmazza a magát az NXT téglát, a rendszer lelkét, 3 szervo motort, egy távolságszenzort, érintésszenzort, fényszenzort és hangszenzort, valamint az érzékelők és motorok téglához kapcsolásához szükséges kábeleket. A 2009-ben megjelent NXT 2.0 ezeken felül további három szenzort tartalmaznak. A boltokban kapható készletekhez jár a Lego saját grafikus programozási nyelve az NXT-G, azonban az NXT tégla firmware-ét lecserélhetjük, hogy egy általunk jobban favorizált magas szintű programozási nyelven, például Javában írhassunk vezérlőszoftvert a robotunk számára. Most pedig vegyük szemügyre a készlet fontosabb elemeit.

3.1. Vezérlőtégla

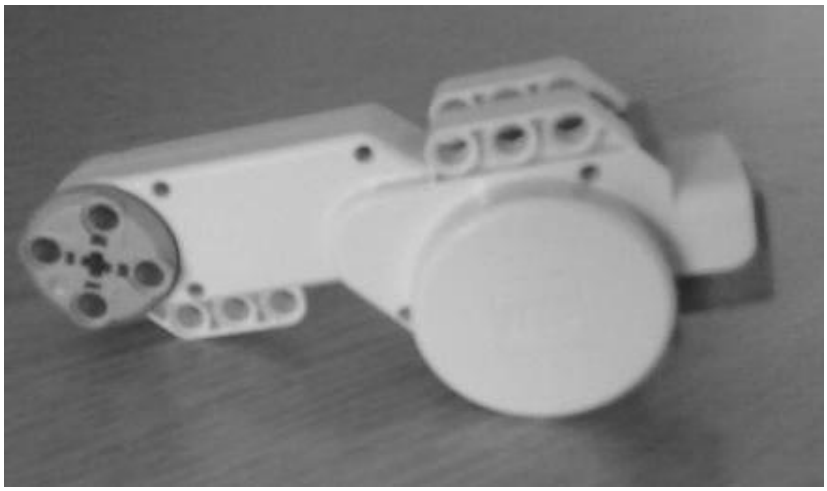
Az intelligens tégla lényegében a Mindstorms robot agya. Ezen tárolódnak a programok, melyek a robot önálló vezérlését teszik lehetővé. Az elődjéhez képest nem csak jobb megjelenésű, de lényegesen ellenállóbb is a sérülésekkel szemben. A téglában egy 32 bites 48 MHz-es ARM processzor található. Rendelkezik 64 Kb RAM-mal amihez a processzor közvetlenül fér hozzá, valamint 256 Kb flash memóriával a programok és adatok tárolására. Ez nem tűnhet soknak elsőre, de az NXT programok nem tartalmaznak kép vagy hangfájlokat, amik a programokban az igazán sok tárterületet elfoglalják.

Nyolc csatlakozó található a téglán. Négy a szenzoroknak, három a motoroknak és egy USB csatlakozó, mellyel a téglát a számítógéphez kötni. Az USB kivételével az összes kábel RJ12-es, nagyon hasonló a RJ11-es telefon vezetékhez. A készlet egy-egy kábelt tartalmaz minden portnak. Az energiát 6 darab 1,5 voltos elem szolgáltatja, de ez kicserélhető a Lego saját Lithium ionos akkumulátorára, ami jól illeszkedik az elemek helyére.

Az NXT téglán négy gomb található, amivel navigálhatunk a tégla firmware-ének menüjében, vagy a saját programunknak is adhatunk ki utasításokat segítségével. Az NXT rendelkezik egy LCD kijelzővel is, ami rengeteget fejlődött az RCX óta. A képernyő 26 mm x 40,6 mm területű és 100 x 64 pixel felbontású. Képes akár 60 Hz-es képernyőfrissítésre is. Lehetővé teszi, hogy a téglán futó programunk futása közben kiírassuk a szenzorok, motorok adatait, kirajzolhassunk egyszerűbb fekete-fehér ábrákat.

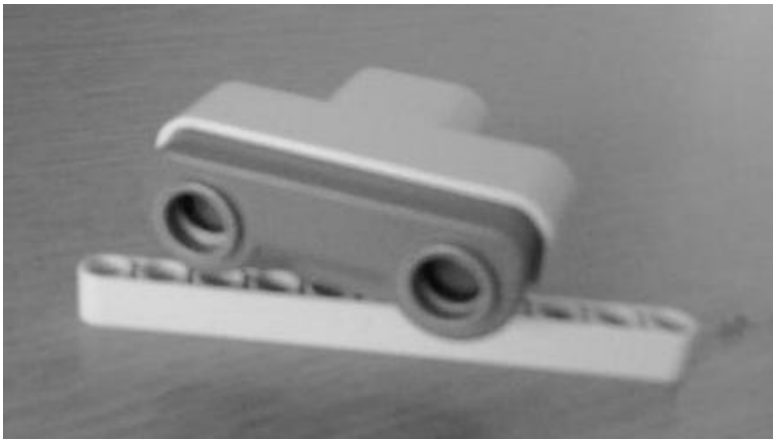
3.2. Motorok

Az NXT készlet három szervo motort tartalmaz. Ezek mondhatni a robot legfontosabb részei, hiszen ha azt akarjuk, hogy a robotunk bármilyen mozgást végezzen, azt a motorok hajtják végre. A szenzoroktól kapott mérési adatok is azt határozzák meg, hogy miként kell a motorokat mozgatni. Az NXT készlet motorainak nagy előnye a beépített fordulatszámmérő. A motor főtengelye akárhányszor körbefordulhat, az számon van tartva és akár több ezer fordulat után vissza tud állni alaphelyzetbe.



3.3. Távolság szenzor

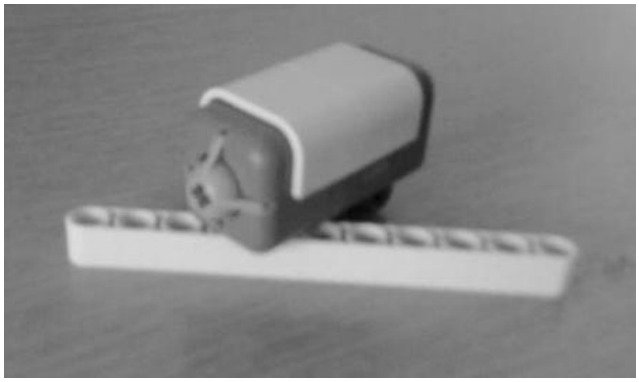
A távolságszenzor vagy ultrahangszenzor a Jávácska kupa szempontjából a legfontosabb szenzor. Ennek segítségével tájékozódik a robotautó a pályán. A szenzor képes érzékelni, ha egy akadály van előtte és meg tudja mérni, hogy az milyen messzire van tőle. A maximális mérhető távolság 255 centiméter, azonban csak 6 és 180 centiméter között mér pontosabban. A mérési eredményekben még ezen a távolságon is nagyjából plusz-mínusz 3 centiméter hiba.



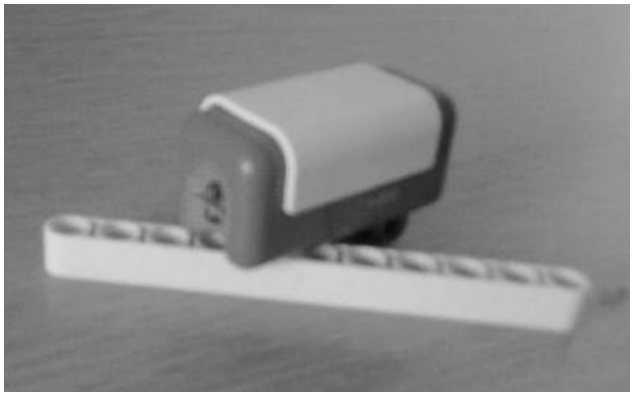
A mérési eredmények annál pontosabbak, minél közelebb van az objektum az érzékelőhöz. A szenzor egy emberi fül számára nem hallható ultrahanghullámot bocsát ki, mely visszaverődik az előtte levő tárgyról. A szenzor érzékeli a visszaérkező hullámot, a visszaérkezés idejéből és a hang terjedési sebességéből ki tudja számolni, hogy az akadály milyen távolságra van. A szenzor a maga előtt levő objektumokat egy nagyjából 30 fokos nyílásszögű kúpon belül érzékeli.

3.4. Egyéb szenzorok

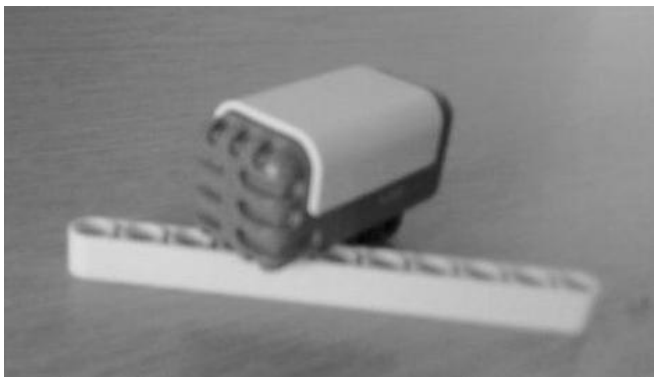
Az NXT készlethez még további szenzorok is tartoznak, amelyek nem lettek beépítve az autóba. Ezek küllemre mind hasonlóak egymáshoz. Az érintés szenzor a legalapvetőbb szenzor a készletben. Egy kapcsoló található az elején. A szenzor azt érzékeli, ha az ehhez a kapcsolóhoz tartozó narancssárga gombot megnyomták, ütközött, vagy elereszteték.



A fényszenzoron egy aprócska lencse található. Az érzékelő az ezen áthaladó fény erősségét képes érzékelni. A szenzor nem csak a látható fény tartományában képes mérni, hanem érzékeli például az infravörös hullámokat is. Mivel a sötétebb színek kisebb erősségű fényt vernek vissza, ezért a szenzor képes megkülönböztetni a sötétebb és világosabb színű objektumokat. Segítségével be lehet állítani a robotot, hogy kövessen egy fekete vonalat vagy például meg lehet akadályozni, hogy lezuhanjon az asztalról.



A hangszenzor mindössze az érzékelt hangok erejét képes mérni decibelben. Bár a Jávácska kupa robotjai még nincsenek felszerelve ezzel a szenzorral, jól lehetne alkalmazni arra például, hogy az autóverseny tapsra vagy startpisztoly lövés hangjára induljon el vagy álljon le.



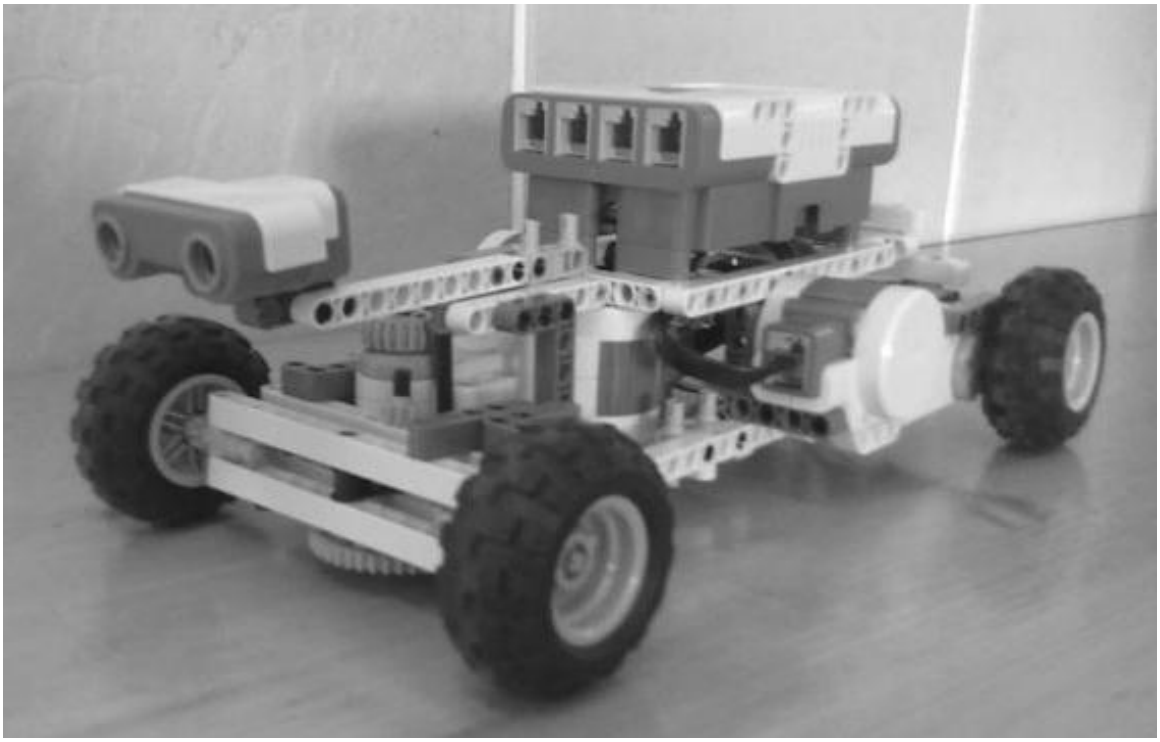
Vannak még további szenzorok is, amiket az alapkészlet nem tartalmaz. Az NXT 2.0 készlet kiegészült egy hőmérséklet szenzorral, iránytű szenzorral valamint gyorsulásmérő. A Legón kívül más cégek is gyártanak szenzorokat a Mindstroms készletet, amik további lehetőségeket nyújtanak a robotok építésére.

3.5. Az autó

Az autó megalkotáskor fontos szempont volt, hogy stabil, jól kormányozható legyen, és nagy sebességet tudjon elérni. Az autó egy téglalap alakú vázra lett építve. Ehhez csatlakozik, a hátsó kerekeket meghajtó két motor, a kormánymű és az első kerekek, a kormányzásért felelős motor, az NXT tégla, valamint az ultrahang szenzor. A kocsi meglehetősen alacsony és széles, így nehezen borul fel. A vezérlő tégla csak nyolc összekapcsoló elemmel van rögzítve a vázon. Ily módon stabilan áll, mégis egy mozdulattal leszedhető, elősegítve a könnyű karbantarthatóságot.

A kormányzást egy motor végzi. A kormánymű kialakításakor ügyelni kellett arra, hogy a kerekek könnyedén elforduljanak, ugyanakkor, ha nem kell fordulnia, lehetőleg ne térjen le a pályáról.

A sebességet egy fogaskerék áttétel növeli. Az autó így háromszor akkora sebességre képes, mintha a kerekeket közvetlenül a motorokra csatlakoznának.



3.5.1. Problémák az autó megalkotása során

Az autót meglepő módon igen nehéz volt rávenni, hogy egyenes vonalon tudjon haladni. Több dologra is figyelni kellett. Az első, mint említettem a kormánymű megfelelő beállítása. A másik részben hardver, részben szoftver probléma. A két hátsó kereket egy-egy motor hajtja meg. Mérési adatok alapján kiderült, hogy bár mindkét motort beállítottuk ugyanarra a sebességre, a mért sebességek időnként véletlenszerű értékkel mégis eltértek a beállítottól. Ez okozta a kocsi egyenes pályától való eltérését. A probléma úgy lett kiküszöbölve, hogy a két tengely össze lett kötve, így kikényszerítve, hogy a két kerék biztosan egyenlő meghajtást kapjon. A kiterjesztett Jávácska kupában sajnos ezt a módszert nem lehet alkalmazni, mivel ott nem egy harmadik motor végzi a kormányzást, hanem a két külön vezérelt motor sebességkülönbsége miatt kanyarodik el a robot.

Az eredeti robotnál gondok adódtak a kanyarodásnál is. Sajnos, ha utasítjuk a robotot, hogy a kanyarodásért felelős motort forgassa el 200 fokkal, akkor az – a leírás szerint is – 198 és 202 fok között is elfordulhat.

Az ultrahang szenzor precizitásával is akadtak problémák. Mivel ultrahang jeleket bocsát ki és azoknak a visszaverődését érzékeli, hibák adódhatnak abból, ha egyszerre két szenzor dolgozik egymás mellett és megzavarják egymást a hullámok. Bár a leírások szerint a LeJOS rendszer szolgáltat metódusokat az ebből adódó hibás mérési eredmények korrigálására, de a tesztek során inkább az derült ki, hogy ezek a metódusok még pontatlanabb adatokat szolgáltatnak.

4. LeJOS

A frissen megvásárolt Lego Mindstorms készlet NXT tégláján alapból van egy firmware, ami a Lego saját grafikus programozási nyelvén az NXT-G nyelven írt programokat képes futtatni. Ez a programnyelv meglehetősen hatékony, jól használható, kiválóan alkalmas arra, hogy a kezdők megtanulhassák a programozás alapjait. Azonban azok számára, akik már jártasságot valamilyen magas szintű programnyelvben, azoknak célszerű az alap firmware-t lecserélni. Az egyik leghatékonyabb eszköz, amivel a robotunkra szoftvert fejleszthetünk, az a leJOS.

A leJOS egy SourceForge project amivel Lego MindStorms robotokra lehet szoftvert fejleszteni Java technológia használatával. Két verziója van. Az egyikkel a korábbi RCX téglára írhatunk programokat, a másikkal az NXT-re. Ebben a dolgozatban az utóbbira fókuszálunk. Ez a leJOS NXJ, ami a következőket tartalmazza.

- Firmware az NXT tégla számára, ami helyettesíti az NXT gyári szoftverét
- A frimware tartalmaz egy kicsi C nyelven írt platform független JVM-et (Java Virtual Machine – Java Virtuális Gép), ami futtatja a programjainkat.
- Tartalmaz egy linkert, ami az általunk írt program osztályokból előállítja a futtatható byte kódot, amit majd a téglára töltünk fel.
- Rendelkezésünkre áll számos PC-s eszköz is melyek segítségével fordíthatjuk a programjainkat és feltölthetjük a téglára. Ezek a programok szintén platform függetlenek, telepíthetők mind Windows, Linux, Macintosh és más egyéb rendszerekre.

A leJOS nagy előnye az NXT-G-vel szemben, hogy robotunk programozása során a Java és általában a magas szintű programozási nyelvek minden szolgáltatását felhasználhatjuk. A leJOS következőket nyújtja a Mindstorms robotprogramozás hatékonyságának növelésére:

- Objektumorientátl programozási nyelv: Programunkat osztályokból építhetjük fel, jelentősen növelve a modularitást, megkönnyítve a karbantartást. Igazi előnyei a később kifejtésre kerülő viselkedésprogramozás során mutatkoznak meg.
- Java típusok használata: Lehet használni a program megírása során a Javaba beépített össze primitív típust. Itt különös jelentősége van a lebegőpontos számok és a lebegőpontos aritmetika használatának. E-nélkül nem lehetne használni az trigonometriai függvényeket, ami elengedhetetlen a navigációhoz.

- Szálak használata: A leJOS megengedi a szálak használatát. Elméletben egyszerre legfeljebb 255 szálat tud kezelni, de természetesen robot programozás során nem lesz szükség ennyire, arról nem is beszélve, hogy a túl sok szál felemészti a memóriát és instabillá teszi a rendszert. A kiterjesztett Jávácska kupában volt alkalmazva a szálkezelés, amikor egyik szálban futott a Bluetooth adatok feldolgozásban, a másikban a viselkedésvezérlés.
- Tömbök: A leJOS lehetővé teszi, hogy nagyobb mennyiségű adatot tömbökben történő tárolását. Lehetséges akár a többdimenziós tömbök használata is. Ez hasznos lehet, ha például a robotunk helyzetét a pályán egy a pályát reprezentáló bitmátrixban tartjuk nyilván.
- A Java kivételkezelése hatékony hibakezelést biztosít.
- A leJOS-ban bár eléggé korlátozott módon, de alkalmazható a rekurzió is. Mivel egy metódus meghívása önmagán belül sok memóriát igényel és az NXT-nek csak 256 Kb állrendelkezésére, ezért csak legfeljebb 10 szint mélységig mehetünk le a hívási láncban. Ez a szám is csökkenhet, ha az önmagát hívó metódusban sok változó van deklarálva.

4.1. Felhasználói interakciók

A leJOS NXJ lehetőséget biztosít arra, hogy a robot és a felhasználó között kommunikáció jöhessen létre. Egyrészt a robot vezérlőprogramja tud üzeneteket küldeni a felhasználónak az NXT tégla beépített LCD kijelzőjén keresztül, másrészt a felhasználó is tudja befolyásolni a program futását a téglán található gombok lenyomásával.

4.1.1. A kijelző programozása

A kijelzőre a program futása során tetszőleges adatokat írathatunk ki. Ezek lehetnek szöveges üzenetek, a program változóinak és konstansainak értékei, a motoroktól és szenzoroktól kapott mérési adatok, vagy egyszerű ábrák. Mivel csak egy kijelző van a készüléken, ezt csak egy statikus osztály reprezentálja, példányosítani nem lehet. Ennek az osztálynak a statikus metódusaival írhatunk vagy rajzolhatunk a képernyőre. Használhatjuk a kijelzőt mind grafikus, mind szöveges módban. Adatok kiírására legalkalmasabb a szöveges mód, ezért ebben a dolgozatban inkább arra fogok részletesebben kitérni.

Szöveges módban a kijelző 16x8 karakternyi területű. Bal felső sarkában van a (0,0) koordinátájú pont a jobb alsóban pedig a (15,7) koordinátájú pont. A kijelzőre írásnak a következőek a fontosabb metódusai:

- void clearDisplay(): Letörli a képernyőt.
- drawString(String str, int x, int y): Kiírja a képernyőre az str paraméterben megadott stringet az x,y paraméterben megadott karakteres képernyő koordinátától kezdve.
- drawInt(int i, int places, int x, int y): Az i paraméterben megadott int számot kiírja a képernyőre az (x,y) paraméterben megadott képernyő koordinátákra. A places opcionális paraméter. A kiírt érték legalább annyi karakterpozíciót foglal el, amennyit ebben a paraméterben megadunk.

4.1.2. Gombok kezelése

Másik módja a felhasználói interakcióknak az NXT téglán található gombok használata. Ezzel a felhasználó futás közben is tudja befolyásolni a robot viselkedését. A téglán négy gomb található:

- ENTER: Nagyobb, négyzet alakú narancssárga gomb.
- ESCAPE: Kisebb téglalap alakú szürke gomb az ENTER alatt.
- LEFT, RIGHT: Az ENTER bal és jobb oldalán található háromszög alakú gombok.

Ezeknek a kezelését a Button osztály metódusai segítségével érhetjük el, melyeket az osztály négy gombnak megfelelő statikus példányain keresztül hívhatjuk meg. A Button osztály fontosabb metódusai:

- void waitForPressAndRelease(): A program (vagy szál) futása addig áll, amíg a metódust meghívó gombot le nem nyomják.
- boolean isPressed(): Igaz értékkel tér vissza, ha a metódust meghívó gomb le van nyomva.

A következő kódrészletben láthatunk példát, mind a gombok, mind a kijelző kezelésére. Ebben a programrészletben a felhasználó állíthatja be a robot sebességét.

```
boolean getAdat = true;
int speed = 500;
//Ciklust indul, amíg a getAdat igaz
while (getAdat) {
     //Letörli a képernyőt és kiírja a bal felső sarokba
     //a sebesség aktuális értékét
     lejos.nxt.LCD.clearDisplay();
     lejos.nxt.LCD.drawString("Speed:\n" + speed, 0, 0);

     //Ha a felhasználó leüti az ENTER gombot, akkor
     //a program true-ra állítja a getAdat-ot
     if (lejos.nxt.Button.ENTER.isPressed()) {
          getAdat = false;
     }

     //Ha a felhasználó leüti a balra vagy jobbra gombot
     //a program csökkenti vagy növeli a speed értékét
     if (lejos.nxt.Button.LEFT.isPressed()) {
          speed -= 50;
     }
     if (lejos.nxt.Button.RIGHT.isPressed()) {
          speed += 50;
     }

     try {
          Thread.sleep(200);
     } catch (Exception e) {
     }
}
```

4.2. Viselkedés programozás

A viselkedés programozás olyan ága a robot programozásnak, ami meglehetősen eltér az korábban megszokott mesterséges intelligencia programozási módszereknek. Segítségével alacsony memóriahasználattal hasonló szintű intelligenciákat hozhatunk létre, mint a rovaroké. A módszert Rodney Brooks professzor fejlesztette ki az MIT Mesterséges Intelligencia Laboratóriumában. A módszert ténylegesen is a rovarok viselkedése ihlette. Azok is rendkívül kevés memóriával rendelkeznek, és mégis jól megállják helyüket a világban. Érzékszerveikkel képesek felfogni környezetükből érkező ingereket és gyorsan tudnak rá reagálni. Hasonlóképpen kell ezt elképzelni robotunk szoftverének írásakor is. A

robot kapja az adatokat vagy ingereket a való világból az szenzorjain keresztül és a motoroknak kiadott parancs formájában reagál rájuk.

Viselkedés programozáskor jóval többről van szó, mint ha-akkor szerkezetek lekódolása. Kézenfekvőnek tűnik ez a megoldás, hiszen könnyedén, különösebb tervezés nélkül megadjuk, hogy ha a robot ezt észleli, akkor ezt kell tennie. Ez kezdetnek, elindulásnak még jó is lehet, de ahogy egyre több képességre akarjuk a robotunkat 'megtanítani' annál kuszább, átláthatatlanabb lesz a kód és a bővítése is rendkívül körülményessé válik.

Ez módszer lényegesen több előretervezést igényel, mint a másik módszer. Pontosan meg kell határozni az egyes reakciókat vagy viselkedésmódokat, és összepárosítani azokat az ingerekkel, aminek hatására bekövetkeznek. A Java objektumorientáltságát kihasználva az egyes viselkedéseket úgynevezett viselkedésosztályokba szervezhetjük. Így egy könnyen megérthető, könnyen átlátható, könnyen karbantartható szerkezetet kapunk. A bármikor hozzáadhatunk vagy elvehetünk viselkedés osztályokat, anélkül, hogy ez a program többi részének működésére bármilyen negatív hatással lenne. A Jávácska kupa robotjának programja is viselkedésekből áll. A következőkben kifejtésre kerül, hogy valósítja meg ezt a leJOS és később, hogy milyen viselkedésekből épül fel a Jávácska robot.

4.2.1. Behavior API

A Behavior API felépítése rendkívül egyszerű, mindössze egy osztályból és interfészből áll, amik a lejos.sumsumption csomagban található. Ezek a Behavior interfész, aminek segítségével meghatározhatjuk a viselkedés, valamint az Arbitrator osztály, ami vezérlésért felel. Ha definiálni akarunk egy viselkedésmódot, létre kell hoznunk egy osztályt, ami megvalósítja az interfészt. Miután minden viselkedést implementáltunk, deklarálnunk kell egy tömböt, ami az egyes viselkedésosztályok példányait tartalmazza, és át kell adni azokat az Arbitrator osztálynak.

4.2.2. Behavior interfész

A Behavior interfészben a következő metódusok találhatóak meg melyeket az egyes viselkedés osztályoknak kell implementálniuk.

- boolean takeControl(): Egy logikai értékkel tér vissza, ami jelzi, hogy aktiválódnia kell-e a viselkedésnek. Például, ha az érintés szenzor azt érzékeli, hogy egy tárgynak ütközött, akkor igaz értékkel tér vissza.
- void action(): Ez a kód fut le, amikor a viselkedés aktívvá válik. Például, amikor a takeControl() metódus érzékeli, hogy a robot ütközik egy objektummal, az action() metódus kiadhatja, hogy a robot forduljon el tőle.
- void supress():A kód a supress() metódusban azonnal leállítja az action() metódusban levő kód futását. Ezen felül használható adatok módosítására mielőtt a viselkedés befejeződik.

Nézzünk egy példát egy Behavior interfészt megvalósító osztályra. A viselkedés akkor lesz aktív, ha az ultrahangszenzor akadályt érzékel a robottól közelebb, mint 50 centiméterre. Ekkor a robot kormányzásért felelős motorja elfordul 200 fokot, vár egy másodpercet, majd újra alapállapotba forgatja a motort.

```
public class Turn implements Behavior {
     //Ultrahang szenzor példány létrhozása
     UltrasonicSensor sonic =
          new UltrasonicSensor(SensorPort.S1);

     public boolean takeControl(){
          return sonic.getDistanc() < 50;
     }

     public void action(){
          try{
               lejos.nxt.Motor.C.rotate(200);
               Thread.sleep(1000);
               lejos.nxt.Motor.C.rotate(-200);
          } catch (Exception e){
          }
     }

     public void supress(){
     }
}
```

4.2.3. Arbitrator osztály

Az Arbitrator osztály elsődleges feladata meghatározni, hogy éppen melyik viselkedést kell aktiválni. Felépítése még egyszerűbb, mint a Behavior interfészé, mivel csak egy metódusa és egy konstruktora van. Ezek a következők:

A konstruktor:

- public Arbitrator(Behaviour[] behaviors): Létrehoz egy Arbitrator objektumot. Paraméterként egy tömböt kell neki átadni, ami a Behavior interfészt megvalósító osztály példányait tartalmazza. Minél magasabb a viselkedés tömbindexe, annál magasabb a prioritása.

Nyilvános metódusa:

- public void start(): Elindítja az vezérlő rendszert. Ekkor az Arbitrator a végéről az eleje felé haladva bejárja a viselkedéseket tartalmazó tömböt, és sorban meghívja azoknak a takeControl() metódusát. Ha valamelyik igaz értékkel tér vissza annak lefuttatja egyszer az action() metódusát, majd újrakezdi a bejárást. Előfordulhat olyan, hogy egyszerre két viselkedés is át akarja venni a vezérlést, mivel azonban a bejárás minden egyes action() hívásnál újra indul, ezért csak az egyikük, a magasabb tömbindexű, azaz a magasabb prioritású fog aktiválódni.

A következő példa kódrészlet létrehoz egy Arbitrator objektumot.

```
//Viselkedéstömb létrehozása
Behavior[] behaviorArray = {
     new ForwardBehavior(),
     new TurnBehavior(),
     new StopBehavior()
};

//Az vezérlőobjektum példányosítása és a vezérlés
//elindítása
Arbitrator arbitrator = new Arbitrator(behavior);
arbitrator.start();
```

Tegyük fel, hogy a konstruktorban példányosított három osztály mindegyike megvalósítja a Behavior interfészt. Ezek közül a legkisebb prioritású a ForwardBehavior objektum a legnagyobb pedig a StopBehavior.

4.3. Motorok kezelése

A lejos.nxt Motor osztály az NXT motorokat reprezentáló osztály. Ezzel az osztállyal kimondottan csak az NXT motorokat lehet kezelni, az RCX-eseket nem, mivel az NXT-be beépített fordulatszámmérő miatt, ezek a motorok egészen más vezérlő metódusokat igényelnek. Az osztály metódusai csak akkor használhatóak, ha legalább egy motor csatlakozik a vezérlő téglához. Az osztály három statikus példánnyal rendelkezik. Ezek Motor.A, Motor.B, Motor.C. Ezeken a példányokon keresztül tudjuk vezérelni az A, B, és C portokra kötött motorokat. Mivel a szenzorportok pontosan ugyanolyan kinézetű porton, szintén RJ12-es kábellel csatlakoznak a téglához, feltételezhetnénk, hogy a motorokat oda is csatlakoztathatjuk, de ez hibákat okozna, mivel a szenzorok és a motorok portjain különböző módon zajlik a kommunikáció.

Az osztály metódusokat szolgáltat a motor vezérlésére és a motorok adatainak lekérdezésére. A motorok beépített fordulatszámmérője, ami nyomon követi a motor tengely körüli fordulatának aktuális szögét fokban. A fordulatszámmérőt lényegében egy rejtett szenzorként is felfoghatjuk. Hogy pontos mérési eredményeket kapjunk, az NXT téglában van külön beépítve egy 8-bites 8 MHz-es AVR processzor 4 Kb flash memóriával és 0.5 Kb RAM-mal.

A motornak két üzemmódja van. Ezek a speedRegulation (sebesség szabályozás) valamint a smoothAcceleration (sima gyorsulás), ami csak akkor működik, ha az előbbi használatban van. Ezek alapértelmezés szerint be vannak kapcsolva. Sebesség szabályozó módban a program a fordulatszámot hasonlítja az eltelt idő és sebesség szorzatához és úgy állítja a motor energiáját, hogy ezek közel legyenek egymáshoz. A sima gyorsulási mód úgy javítja a sebességszabályozást, hogy figyelembe vegye a gyorsulási időt. Ezeket a módokat ki és be lehet kapcsolni a regulateSpeed() és a smoothAccleration() metódusokkal.

4.3.1. Motor irányítása

- forward(): Elindítja a motort és előrefele pörgeti.
- backward(): Visszafele pörgetve indítja el a motort.
- changeDirection(): Megváltoztatja a motor forgásirányát.
- stop(): Leállítja a motort.

- flt(): Megszünteti a motor áramellátását, de nem állítja meg azonnal, hanem hagyja, hogy szép lassan magától álljon meg.
- rotate(angle, boolean immediateReturn): Elforgatja a motort a paraméterben megadott szöggel. Az immediateReturn opcionális paraméter. Ha az értéke igaz, akkor a metódus azonnal visszatér.
- rotateTo(angle, boolean immediateReturn): A paraméterben megadott szögállásig forgatja a motort. Az immediateReturn opcionális paraméter. Ha az értéke igaz, akkor a metódus azonnal visszatér.
- setSpeed(int speed): Bállítja a motor sebességét fok/másodpercben. A motor által elérhető maximális sebesség 900 fok/másodperc, de csak teljesen feltöltött elemek esetén.

4.3.2. Motor adatainak lekérdezése

- int getSpeed(): Visszaadja a motor beállított sebességét fok/másodpercben.
- int getActualSpeed(): A motor sebessége nemcsak a programunk által beállított értéktől függ, hanem komolyan befolyásolja az elemek töltöttsége is. Ez a metódus visszaadja a motor valós sebességét fok/másodpercben. Az érték 100 milliszekundumonként számolja ki a fordulatszámmérőt kezelő az NXT téglába beépített másodlagos processzor és 10 szög/s pontossággal kapjuk meg. Arra használjuk, hogy észleljük, ha csökken a motorsebesség.
- int getMode(): Lekérdezhetjük a motor aktuális állapotát. Ez egy kódszámot ad vissza melynek értelmezése:
    - 1, ha előrefele pörög;
    - 2, ha hátrafele pörög;
    - 3, ha áll;
    - 4, ha áramellátás nélkül még pörög a motor, más szóval lebeg.

Vannak más metódusok is a motor állapotának lekérdezéséhez, melyek logikai értéket adnak vissza:

- boolean isForward()
- boolean isBackward()
- boolean isStopped()

- boolean isFloating()

A következő példakód beállítja a robotot meghajtó motorok sebességét és elindítja őket. Ezek után egy ciklusban másodpercenként figyeli a fordulatszámot. Ha az A motor átlépte az ötvenezret, akkor kilép a ciklusból, leállítja a motorokat és kiírja a kijelzőre mindekét fordulatszámmérő állását.

```
//Meghajtó motorok sebességének beállítása…
lejos.nxt.Motor.A.setSpeed(700);
lejos.nxt.Motor.B.setSpeed(700);

//… és elindítása
lejos.nxt.Motor.A.start();
lejos.nxt.Motor.B.start();

//ciklus indítása
while(lejos.nxt.Motor.A.getTachoCount()<=50000){
     try{
          Thread.sleep(1000);
     } catch (Exception e){
     }
}
lejos.nxt.LCD.drawString(”A: ” +
lejos.nxt.Motor.A.getTachoCount()+ ”\n”);
lejos.nxt.LCD.drawString(”B: ” +
lejos.nxt.Motor.B.getTachoCount()+ ”\n”);
```

4.4. Az érzékelő programozása

Ha a távolságszenzort vagy bármilyen más érzékelőt szeretnénk programozni, akkor a szenzornak megfelelő osztályt kell példányosítanunk és annak a metódusait kell meghívnunk. A leJOS NXJ rendszerben mind a négy NXT szenzorfajtának, valamint számos már gyártó által forgalmazott szenzoroknak is megvan a maga reprezentáló osztálya. A szenzorok csak akkor működnek, ha téglához csatlakoztattuk valamelyik szenzor porton keresztül. Hogy a szenzort használhassuk, az azt reprezentáló objektumnak tudnia kell, hogy melyik portra van kötve. Az érzékelőknek a portjait a SensorPort osztály reprezentálja. A szenzor osztály példányosításakor ennek az osztálynak egyik statikus példányát kell átadnunk a konstruktor paramétereként. Ezek a példányok: S1, S2, S3, S4. Ultrahangos szenzort a következőképp példányosíthatunk:

- UltrasonicSensor(Port aSensorPort)

A szenzornak két üzemmódja van, folyamatos (continous, az alapértelmezett) és a ping mód. Folyamatos módban a szenzor olyan gyakorisággal küldi az ultrahang jeleket (más szóval pingel), amilyen gyakran csak tudja, míg ping módban csak akkor küld ultrahang jelet, hogyha erre a program utasítást ad. Elméletileg létezik még egy harmadik módja is a szenzornak, az úgynevezett capture mód. Ebben az üzemmódba elvileg a program kiküszöböli azokat a hibákat, amiket egy másik közel üzemelő távolságszenzor által kibocsátott jelek okoznának, de az elvégzett tesztek azt mutatták, hogy ilyenkor a szenzor meglehetősen rapszodikus mérési eredményeket ad. Ennek ellenére nem capture módban, két egymás mellett működő szenzor úgy tűnt, hogy nem zavarja meg egymást és helyes eredményeket szolgáltat.

Az UltrasonicSensor osztály legfontosabb metódusai a következők:

- int getDistance(): Folyamatos módban visszaadja a szenzor által legkésőbb mért távolságértéket. A visszatérési érték centiméterben értendő. Ha a szenzor nem észlelt visszhangot, akkor 255-öt kapunk eredményül. Mivel a szenzor két mérése között eltelik úgy 20-30 milliszekundum, célszerű a metódus két meghívása között ennyit várni. Egy ultrahanghullám a maximális érték mérése esetén 510 centiméter utat tesz meg 15 ms idő alatt.
- void ping(): Ez a metódus ping módba állítja a szenzort, és kiküld egy jelet. A beérkező visszhangok közül legfeljebb 8-at rögzít.
- int getDistances(int[] distances): A ping metódus által észlelt visszhangokat lehet lekérdezni ezzel a metódussal. A metódusnak paraméterként egy akkora int tömböt kell átadni, amennyi mérési adatot szeretnénk megkapni, de legfeljebb 8 eleműt. A metódus lefutása után a tömb tartalmazni fogja a mérési eredményeket. Mivel a mérés időbe telik célszerű a ping és a getDistances hívása közé célszerű beszúrni egy 20-30 másodperces késleltetést. A várakozást egyik metódus sem tartalmazza. A lekérdezés idő előtti meghívása hibás mérési eredményeket okozhat, de az is lehet hogy egyszerűen nem kapunk vissza semmit. A ping után meg lehet hívni a sima getDistance metódust is. Ekkor az első visszhang értékét adja vissza.
- int continous(): A ping() metódus meghívása kikapcsolja az alapértelmezett folyamatos módot. Hogy ezt újra aktiváljuk ezt a metódust kell meghívni:

A következő példakódban létrehozunk egy ultrahang szenzor példányt és elindítunk egy motort. Ha a szenzor 50 centiméteren belül akadályt észlel, akkor leállítja a motort.

```
//Az első portra csatlakoztatott szenzornak példány
//létrehozása
UltrasonicSensor sonic =
     new UltrasonicSensor(SensorPort.S1);
lejos.nxt.Motor.A.forward(); //Motor elindítása

//Ciklus indítása, melyben 30 ms-ként figyeli a mért
//távolságot
while(true){
     //Ha a távolság kisebb mint 50 cm kilép a ciklusból
     if(sonic.getDistance<50){
          break;
     }
     try{
          Thread.sleep(30);
     } catch (Exception e){
     }

}
//A ciklus lefutása után leállítja a motort.
lejos.nxt.Motor.A.stop();
```

4.5. Viselkedés osztályok a Jávácska kupa robotjában

Az irányítást négy fő viselkedésforma megadásával valósult meg. Mindegyik viselkedés külön osztályban lett definiálva. A főosztály main() metódusában a felhasználói interakció van implementálva. A program elindulása után a felhasználó az NXT tégla gombjai segítségével beállíthatja a robot meghajtó motorjainak sebességét és egy korrigálási számot. Ezután létrehozza a viselkedésosztályok példányaiból a viselkedéstömböt, amit átadva Arbitrator osztály konstruktorának létrehoz egy vezérlő objektumot. Végül meghívja az Arbitrator példány start() metódusát elindítva a szabályozási folyamatot.

Nézzük, hogy az egyes viselkedések milyen szerepet töltenek be a modellben és mi módon vannak implementálva a Behavior interfész metódusai. Elöljáróban megjegyezném, hogy a supress() metódus törzse mindegyik osztály esetében üres.

4.5.1 Forward Behaviour

Az Arbitratornak átadandó tömbben az első elem, tehát ennek a legkisebb a prioritása. Bármelyik másik viselkedés takeControl()-ja true-t ad, az leállítja a Forward futását. Metódusai:

- takeControl(): Mindig igaz értékkel tér vissza.
- action(): Ez egyszerűen beállítja a meghajtó motorok (Motor.A és Motor.B) sebességét a felhasználó által megadottra és elindítja őket.

4.5.2. Turn Behaviour

A legösszetettebb viselkedés felelős a kanyarodás elvégzéséért.

A feltétel, melynek hatására aktiválódik a viselkedés, hogy a jármű elég közel kerüljön a falhoz. Ez a távolság függ a sebességtől. Metódusai:

- takeControl(): Iagz értékkel tér vissza, ha az ultrahang-szenzor által mért távolság kisebb a motor forgási sebességének 15 %-ától.
- action(): Első lépésben 400 szög/másodperces sebességre lassít a motor. A tesztek azt mutatták, hogy nagyobb sebességű fordulással a kocsi rendkívül pontatlanul veszi be a kanyart. 400 volt az a sebesség, aminél a kanyarodás NXT rendszer hibáiból adódó pontatlanságai elfogadhatóak, de a kocsi viszonylag gyors marad. A második lépésben a kanyarodásért felelős motor (Motor.C) forog el 200 fokkal. Egy 5:1 arányú fogaskerékáttét miatt a kerekek csak 40 fokot fordulnak. Addig marad ebben az állapotban, amíg az ultrahang szenzor 170 centimétertől közelebb érzékel valamit. Ezután a motor visszaáll alaphelyzetbe. Mivel a metódus csak +/- 1-2 fokos hibával forgatja a motort, ezért a kanyar bevétele után korrigálásra van szükség. Ha a fordulatszámmérő visszaállás után nem nullán áll, akkor a C motort elforgatja a program elindítása után bekért korrigálási szöggel pozitív vagy negatív irányba, attól függően, hogy a fordulatszámmérő állása kisebb vagy nagyobb nullától.

4.5.3. Stop Behaviour

A legnagyobb prioritású viselkedés, ami leállítja a kocsit. A felhasználótól függ, hogy leállítja-e a program futását.

- takeControl(): Igaz értékkel tér vissza, ha a felhasználó lenyomja az NXT tégla ESCAPE gombját.
- action(): Leállítja a motorokat és kilép a programból.

4.5.4. Stuck Behaviuor

A TurnBehavior elméletben megakadályozná, hogy a kocsi falnak ütközzön, de jelentkezhetnek váratlan akadályok és a NXT rendszernek is vannak pontatlanságai. Megeshet, hogy a kocsit mégis megállítja valami, de a motorok pörögnének tovább. Ez leállítja őket, ha mégis falnak ütközne.

- takeControl(): Igaz értékkel tér vissza, ha mindkét meghajtó motor valós sebessége (getActualSpeed() által visszaadott érték) 50 alá csökken, és a szenzor is 10 cm-nél kisebb távolságot érzékel.
- action(): Leállítja a motorokat és kilép a programból.

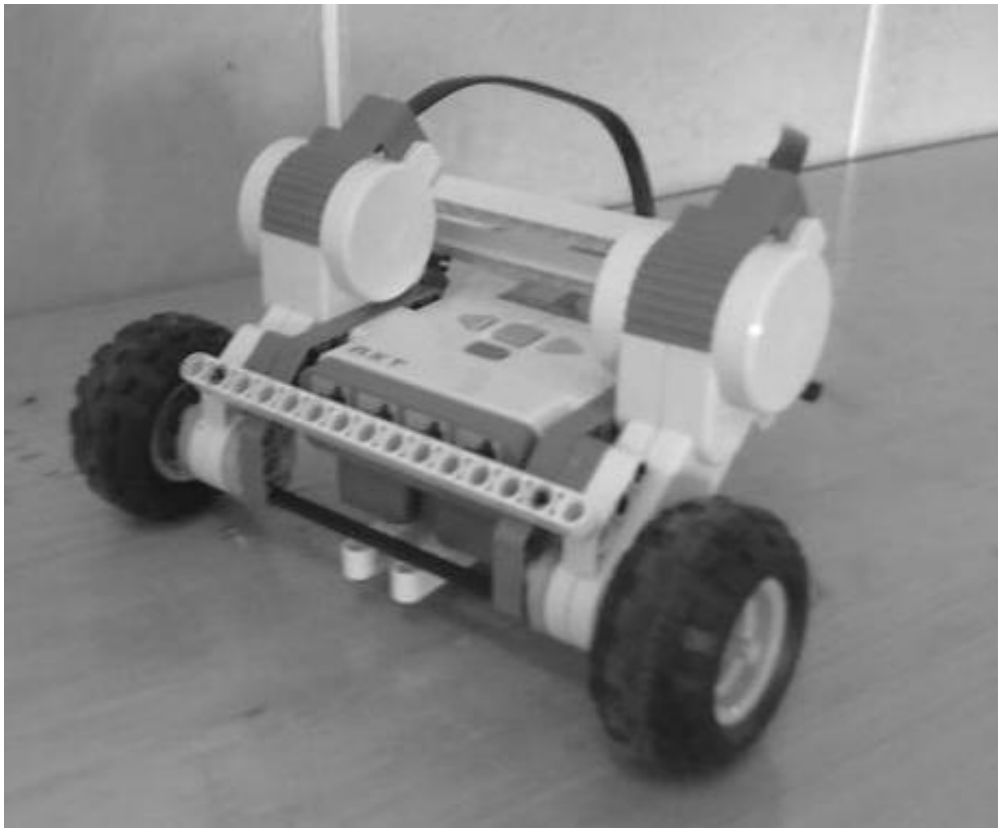
Születtek olyan megoldások is erre az esetre, amelynél nem áll le a kocsi, hanem eltolat az akadálytól és némiképp elkanyarodva folytatja útját. Olyan verziót azonban még nem láttam, ami megfelelő irányban halad tovább a pályán az ütközés után. Természetesen ez nem jelenti azt, hogy nem lehet megvalósítani, vagy esetleg más csapatok a Jávácska kupén nem valósították már meg időközben.

4.6. TachoPilot osztály

A Pilot interfész kimondottan robot autók vezérlésére készült. Az osztály amelyik ennek ezt az interfészt megvalósítja az lehetővé teszi, hogy navigálja a robotunkat a pályán. A leJOS rendszerben az egyik eleve létező ilyen osztály a TachoPilot. Ez az osztály elrejti a programozó elől a motorok kezelését. Használatakor nem szükséges sok metódushívással beállítani több motor sebességét, amiket azután a megfelelő irányba és megfelelő szöggel forgatunk. Egyetlen metódushívással elérhetjük, hogy az autó haladjon előre vagy hátra, egyenes vonalon vagy körpályán, vagy forduljon új irányba. Ez az osztály az alap Jávácska kupa robotjában nem szerepel, de a Kiterjesztett verzióban ez végzi az irányítást.

Meg kell jegyezni, hogy ez az osztály másfajta felépítést igényel, mint az előző robot. Használatához a robotnak két külön kezelt motorral kell rendelkeznie, melyekkel

differenciálműves kormányzást valósítanak meg. Példányosításkor nem csak a Motor osztály megfelelő statikus példányait kell átadnunk, hanem információkat is a robot felépítéséről.



4.6.1. Konstruktor

- public Pilot(float wheelDiameter, float trackWidth, Motor leftMotor, Motor rightMotor[, boolean reverse])

  Paraméterei:

  wheelDiameter: A kerék átmérője, bármilyen elfogadott mértékegységben.

  trackWidth: A távolság a jobb és a bal kerék középpontja között. Ugyanabban a mértékegységben, mint a kerékátmérő.

  leftMotor, rightMotor: A bal és jobboldali motorok példányai.

  reverse: Opcionális paraméter. Ha igaz, az NXT robot előrehalad, amikor a motorok hátrafelé forognak.

  Létezik a konstruktornak olyan változata is, amelyben a két kerék átmérője különböző lehet.

4.6.2. Fontosabb metódusai

- forward(): Előrefele mozgatja a robotot. A kocsi mindaddig menni fog, amíg a stop() metódus nincs hívva.
- backward(): Hátrafele mozgatja a robotot. A kocsi mindaddig menni fog amíg a stop() metódus nincs hívva.
- rotate(int angle, boolean immediateReturn): Hatására a robot elkanyarodik az angle paraméterben megadott szöggel. Ha a szög pozitív, akkor balra fordul, ha negatív jobbra. Az immediateReturn opcionális paraméter, ha igaz a metódus azonnal visszatér, egyébként a metódus csak akkor tér vissza, ha a kocsi elérte a megadott szöget.
- setSpeed(int speed): Beállítja mindkét motor sebességét a paraméterként megadott sebességre, valamint bekapcsolja a sebességszabályozást.
- steer(int turnRate, int angle, boolean immediateReturn): Körpályán mozgatja a robotot, a turnRate paraméterben megadott fordulási arány szerint. Ha pozitív a pálya középpontja a robottól balra van. Ez határozza meg százalékosan, hogy a belső kerék mennyivel forogjon lassabban, mint a külső. Ha meg van adva az angle opcionális paraméter, akkor a robot az ebben a paraméterben megadott szögig mozog. Ha ez a szög negatív, akkor hátrafele halad. Az immediateReturn szintén opcionális paraméter, ha igaz a metódus azonnal visszatér.
- travel(float distance, boolean immediateReturn): Mozgatja a robotot, a distance paraméterben megadott távolságra. Pozitív érték esetén előre mozog, negatívnál hátra. Az immediateReturn opcionális paraméter, ha igaz a metódus azonnal visszatér.
- stop(): Megállítja a robotot.

A következő példában létrehozunk egy TachoPilot példányt. A Pilot által vezérelt robot kerekeinek átmérője 23 mm, a kerekeinek távolsága 170 mm. A definiálás után a robot egy négyzet alakú pályán fog haladni, amíg a felhasználó le nem üti az ESCAPE gombot a téglán. A példában a rotate() parancsal kanyarodik az autó, így egy helyben fordul meg. Ha a steer() metódus hívásával van a kanyarodás elvégezve, akkor a robot körpályán fog kanyarodni.

```
private TachoPilot pilot;
pilot = new TachoPilot
     (23.0,23.0,170.0,MotorA,Motor.B,false);

public static void main(){
     while(!lejos.nxt.Button.ESCAPE.isPressed()){
          pilot.travel(100);
          pilot.rotate(90);
     }
}
```

5. Kiterjesztett Jávácska kupa

Az eddigi tesztek során kiderült, hogy az NXT készlet által nyújtott szenzorok nem túl pontosak. A Lego Mindstorms leírásban is fellelhető, hogy némi hiba lehet a mért értékekben. Ez adta az ötletet, hogy a szenzort valami hatékonyabbal kellene helyettesíteni. A választás a mobiltelefonokra esett. A szenzor helyére egy kamerával rendelkező mobiltelefon lett helyezve. A kamera által látott képet egy szoftver dolgozza fel és a kapott eredmények alapján parancsokat küld a vezérlőtéglának Bluetooth kapcsolaton keresztül, hogy fordulnia kell-e vagy egyenesen menni.

5.1. Bluetooth kommunikáció

A Bluetooth technológia vezeték nélküli kommunikációt valósít meg. Segítségével egészen kis hálózatokat, úgynevezett piconet hálózatokat hozhatunk létre. Hasonló frekvencián működik, mint a WiFi, ami kb 2,4 GHz, de fontos ne keverjük vele. A WiFi egészen más jellegű és jóval nagyobb hálózatok létrehozására alkalmas. A Bluetooth eszközöket aszerint, hogy milyen nagy a hatósugaruk három kategóriába sorolhatjuk.

- Class 3: Maximális távolság, amit át tud hidalni 1 méter.
- Class 2: Maximális távolság 10 méter.
- Class 1: Ez már képes összekötni egymástól 100 méterre lévő eszközöket is.

A Bluetooth technológiát leginkább kis méretű adatok küldésére célszerű használni, mivel meglehetősen lassú, mindössze 460 KBit/s adatátviteli sebességre képes. Ez azonban ideális legfeljebb egy-két bájt méretű parancsok küldésére, amivel tökéletesen lehet vezérelni a robotot.

A vezérlőtégla képes a Bluetooth technológia segítségével a vezeték nélkül kommunikálni a számítógéppel, mobiltelefonnal, PDA-val vagy más egyéb Bluetooth eszközzel, akár egy másik NXT téglával is. Ezáltal lehetőségek végtelen tárháza nyílik meg a robotprogramozó előtt. Bluetoothon keresztül egyszerűen csak feltölthetjük vezérlőprogramjainkat is a téglára, de egy számítógépes program is küldhet parancsokat a robotnak, megvalósítható távirányítás egy mobiltelefonon keresztül, a robot tájékozódása GPS adatok alapján. Azt is elérhetjük Bluetooth segítségével, hogy a robot olyan hatalmas adatbázisokhoz férhessen hozzá, ami soha nem férne el a memóriájában.

A leJOS technológia képes kezelni a Bluetooth kapcsolatokat és végrehajtani az adatcserét más eszközökkel. Ahhoz, hogy a leJOS segítségével kezelni tudjuk a Bluetooth kommunikációt, a következő csomagokat kell a programunkba importálnunk:

- java.io.*
- javax.bluetooth.*
- lejos.nxt.comm.*
- lejos.nxt.remote.*
- lejos.devices.*

Kapcsolódáshoz és a Bluetooth eszköz kezeléséhez a programunkban a következő lépéseket kell végrehajtani:

1. Megtalálni a Bluetooth eszközt
2. Csatlakozni az eszközhöz
3. Adatcsere az NXT tégla és az eszköz között
4. Figyelni a kapcsolatot

5.1.1. Eszközök keresése

Mielőtt csatlakozni szeretnénk egy másik NXT téglához vagy egyéb Bluetooth eszközhöz, szükséges tudnunk, hogy milyen eszközhöz szeretnénk csatlakozni, és hogy elérhető-e ez az eszköz. A vezérlőtégla a memóriájában tárolja az információt azokról az eszközökről melyekhez már korábban csatlakozott. A leJOS frimware Bluetooth menüjéből megtudhatjuk, hogy melyek ezek az eszközök, valamint ugyanitt csatlakoztathatunk is új eszközöket. Ezeket az információkat általunk írt programból is lekérdezhetjük. A lejos.nxt.comm csomag Bluetooth osztálya tartalmaz egy statikus metódust erre a célra.

- Vector getKnownDevicesList(): Visszaad egy vektort, mely tartalmazza az összes a téglával már párosított eszközt.

Ha egy még nem ismert eszközhöz akarunk csatlakozni, akkor a következő metódust kell használnunk.

- Vector inquire(int max, int timeout, byte[] cod):Ez szintén egy vektorban adja vissza a tégla környezetében észlelt, még nem párosított eszközöket. Ha a vizsgálat sikerrel járt, akkor érdemes a kapott vektort bejárnunk és a fellelt eszközöket hozzáadni a NXT tégla listájához. Paraméterei:
  max: A felderítendő készülékek maximális száma

timeout: Az maximális idő, amíg a keresés fut, ha nem talál a közelben készülékeket.

cod: A kapcsolódáskor használt pin kód számjegyeit tartalmazó byte tömb.

5.1.2. Kapcsolódás az eszközhöz

Ha az eszköz hozzálett adva a téglához, akkor következő lépésként csatlakozni kell hozzá, hogy kommunikálni lehessen vele. Első lépésként deklarálnunk kell egy RemoteDevice objektumot. Ez az objektum nem konstruktorhívással jön létre, hanem a Bluetooth osztály következő statikus metódusa adja vissza:

- RemoteDevice getKnownDevice(String devname): Vissza adja egy RemoteDevice objektum segítségével reprezentálva azt a már kapcsolódott és az NXT tégla memóriájában letárolt Bluetooth eszközt, aminek a neve megegyezik a paraméterben megadottal.
- BTConnection connect(RemoteDevice dev): Ha létezik az eszköz, akkor ezzel a metódussal csatlakozni lehet hozzá. Ha sikeres a csatlakozás, akkor egy Bluetooth kapcsolatot reprezentáló BTConnection objektumot ad vissza. A metódusnak ezen verziója a távoli eszköz pin kódját használja a csatlakozáskor.
- BTConnection connect(String target, int mode, byte[] pin): Ebben a három paraméteres metódusban pontosabban lehet definiálni a kapcsolatot. Paraméterei:
  target: Egy Stringben adjuk át a távoli eszköz nevét vagy címét.
  mode: Az I/O kommunikáció módjának a kódját egy int paraméterben. A BTConnection leszármazott osztályában az NXTConnectionban előre definiált konstansok vannak a mód megadására.
  pin: Egy byte tömbben kell átadni a távoli eszköz pin kódját.

5.1.3. Adatcsere az NXT tégla és az eszköz között

Két módja létezik az adatcserének. Az egyik, hogy a Bluetooth kapcsolat I/O módját beállítjuk NXT kapcsolatok fogadására, majd a kapcsolatból olvasunk adatokat és kapcsolatba írunk.

A másik a InputStream és OutputStream objektumok használata. Első lépésként a connect() metódus által visszaadott BTConnection objektumból meg kell nyitni az I/O Stream objektumokat a következő metódusok segítségével.

- DataInputStream openDataInputStream(): Visszaadja a metódust meghívó BTConnection objektumhoz tartozó DataInputStream példányt.
- DataOutputStream openDataOutputStream(): Visszaadja a metódust meghívó BTConnection objektumhoz tartozó DataOutputStream példányt.

Ezek után az I/O Stream objektumok író és olvasó metódusaival küldhetünk és fogadhatunk adatokat a távoli eszköztől.

5.1.4. Kapcsolat figyelés

A Bluetooth kapcsolatok figyeléséhez a következő a Bluetooth osztály következő statikus metódusát kell meghívni:

- BTConnection waitForConnection(int timeout, int mode, byte[] pin): Addig vár, amíg egy távoli eszköz nem csatlakozik. Sikeres csatlakozás esetén a metódus egy Bluetooth kapcsolatot reprezentáló BTConnection objektummal tér vissza. Paraméterei mind opcionálisak.

  timeout: Ebben a paraméterben megadhatjuk, hogy legfeljebb mennyi ideig várjon a metódus a csatlakozásra, 0 esetén örökké vár.

  mode: Ebben a paraméterben az I/O módot lehet beállítani, amit a kapcsolat használni fog.

  pin: Ebben a byte-tömbben lehet átadni a metódusnak a távoli eszköz pin kódját, amit kapcsolat kialakítása során használni fog. Ha nem adjuk meg akkor az alapértelmezett pin-kódot használja.

Amint a távoli eszköz csatlakozott az NXT téglához, máris megkezdődhet az adatátvitel InputStream és OutputStream objektumok használatával.

A következő példakód a Bluetooth kommunikáció egy lehetséges módját mutatja be. A programrészlet először egy bájkódban beállítja a pin kódot, majd ezzel a kóddal vár egy eszköz csatlakozására. Ha a csatlakozás sikeres, létrehozza az InputStream objektumot és elkezdi egy végtelen ciklusban beolvasni a távoli eszköz által küldött adatokat.

```
//Pin kód beállítása és létrehozza egy NXTConnection
//objektumot, ami leszármazottja a BTConnection osztálynal
byte[] pin = { 0, 0, 0, 0 };
NXTConnection connection =
     Bluetooth.waitForConnection(0,
     NXTConnection.RAW, pin);

//Az InputStream objektum létrehozása
InputStream dataIn = connection.openInputStream();

//Egy ciklusban az adatok beolvasása
while (true) {
     int b = dataIn.read();
     /*
     Ide jön a b változóba beolvasott adatok feldolgozása
     */
}
```

5.2. Mobiltelefonos vezérlés

A Bluetooth kapcsolat jóvoltából az NXT tégla képes egy Bluetooth kommunikációra képes mobiltelefonokkal is. Ennek segítségével mobil készülékre írhatunk olyan programot mely segítségével távirányítással vezérelhetjük a robotot.

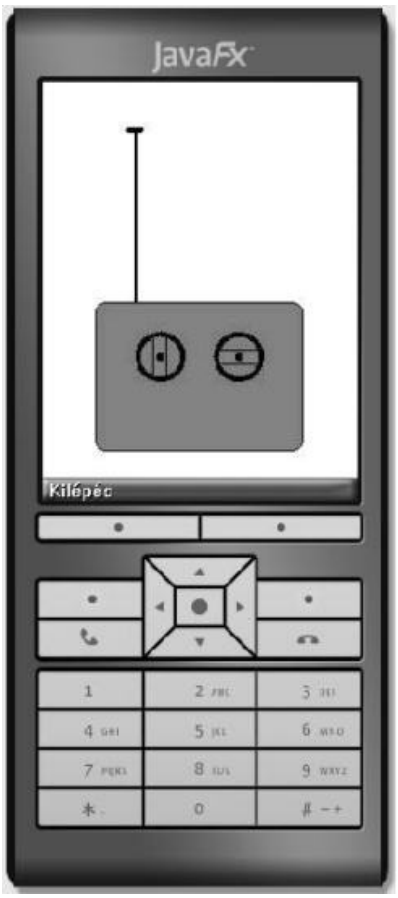
A távirányítóhoz két program szükséges. Egyik a telefonra, mely Bluetooth kapcsolaton keresztül küldi a parancsokat a robotnak a felhasználói interakciók alapján. A másik az NXT téglára, mely fogadja a telefonról érkező adatokat és azok alapján utasításokat ad ki a robot motorjainak.

5.2.1. LegoRemote

Először vegyük szemügyre a mobiltelefonra készült programot. A szoftver egy Java ME technológiával készült mobil alkalmazás. A program futtatásához olyan mobiltelefonra van szükségünk, ami képes kezelni a MIDP 2.0 készülék profilt és a CLDC 1.1 eszköz konfigurációt valamint lehetséges rajta a Bluetooth kommunikáció. A program három fő osztályból áll:

- BTClient

- LegoRemoteMIDlet
- LegoRemoteCanvas



BTClient: Ez az osztály kezeli a Bluetooth kapcsolatot. Tartalmaz metódusokat Bluetooth eszközök keresésére, azok szolgáltatásainak fellelésére, az eszközhöz való csatlakozására és adatok küldésére.

LegoRemoteMIDlet: Ez a programnak a MIDlet osztálya. Minden mobiltelefonos alkalmazásnak a MIDlet osztály a fő része. Az alkalmazás elindításakor példányosítja a LegoRemoteCanvas osztályt és beállítja, hogy a kijelzőn a vászon által rajzolt kép jelenjen meg. Beállít a Canvasnak még egy 'Kilépés' gombot és egy parancs figyelőt. Onnantól kezdve a Canvas metódusai kapják meg a vezérlést.

LegoRemoteCanvas: Alapvetően a telefon kijelzőjén mindenkor megjelenő képet reprezentáló osztály. Ez az osztály kezeli a felhasználói interakciókat is. Az osztály implementálja a Runnable interfészt is, azaz van egy run() metódusa, amely külön szálban fut. Az osztály legfontosabb metódusai:

- Konstruktor: Paramétere egy MIDlet példány. Példányosítja a BTClient osztályt. Ezek után betölti a képeket, melyekből a háttér összeáll majd. Végül elindítja a szál futását.
- keyPressed(): Ez a metódus akkor aktiválódik, ha a felhasználó lenyomja a telefon egyik gombját. Ha lenyomják a 2, 4, 6, 8 vagy a fel, balra, jobbra, le gombokat küld egy bájtot Bluetoothon keresztül az NXT téglának.
- keyReleased(): Ez a metódus akkor aktiválódik, ha egy lenyomott gombot elengednek. Ha a keyPressed() metódusnál felsorolt gombokat elengedi a felhasználó küld egy bájtot Bluetooth-on keresztül az NXT téglának.

- paint(): Kirajzolja a vásznat. Hogy mit rajzol vagy ír ki grafikusan a telefon kijelzőjére, az attól függ, hogy hol tart éppen a vezérlőtégla megkeresésében illetve milyen gombot ütött le a felhasználó.
- run(): Első lépésben kirajzolja az indítóképernyőt, majd elkezdi keresni a közelben levő Bluetooth eszközöket. Ha talál készülékeket, akkor kikeresi közülük az NXT téglát. Ha nem talál vezérlőtéglát, akkor ezt a program kiírja, ha igen akkor megpróbál csatlakozni hozzá. Ha a csatlakozás sikertelen értesíti róla a felhasználót, ha sikeres, akkor megjelenik a képernyőn a távirányító és onnan kezdve az alkalmazás a felhasználó interakcióira vár.

5.2.2. RemoteControl

A távirányítás megvalósításához elengedhetetlen egy másik program megírása is a vezérlőtégla számára, ami feldolgozza a telefonról érkező adatokat. A szoftver egészen egyszerű, mindössze egy főosztályból áll. A telefon egy bájtot küld, de ez is elég a robot távirányításához. Az osztálynak egy boolean típusú adattagja (running) van, ami azt mondja meg, hogy a program főciklusának kell-e még futnia. Alapértelmezett értéke true.
A program először egy programszálat hoz létre. Ennek a run() metódusának kezdetén, négyelemű byte tömbben beállítja a vezérlőtégla pin kódját. Ezek után a program várja, hogy ezzel a pin kóddal eszköz csatlakozzon a téglához. Ha az összeköttetés létrejött, kiírja az NXT tégla kijelzőjére, majd a Connection objektumból létrehozza az InputStream-et és várja a telefontól érkező byte-okat.
Ha minden előkészült elindít egy while ciklust, ami addig fut, amíg a running adattag értéke igaz. A ciklus elején egy int változóba (b) beolvas egy bájtot az InputStreamről. Ennek az értékétől függ, hogy mit kell a robotnak csinálnia.

- 0: Akkor küldi ezt a telefont, ha a felhasználó leüti a 2-es vagy a FEL gombot a készüléken. Ebben az esetben a robot beállítja a motorok sebességét 900 szög/másodpercre.
- 1: Ha a felhasználó elengedi a 2-es, 8-as, FEL és LE gombokat akkor a telefon küld egyet. Ennek a jelnek a fogadása esetén a program leállítja a motorokat.
- 2: A telefon kettes értéket küld, ha a felhasználó lenyomja a négyes vagy a BALRA gombot. Ebben az esetben a robot elforgatja a kanyarodásért felelős

motorját 200 fokkal. Ez egy fogaskerék áttételnek köszönhetően a robot 40 fokos elkanyarodását eredményezi balra.

- 3: Akkor kap a program hármas értéket a telefontól, ha a falhasználó lenyomja a hatos vagy a JOBBRA gombot. Ekkor a robot -200 fokkal forgatja el a kanyarodó motort.
- 4: Ha a felhasználó elengedi a 4-es, 6-os, BALRA, vagy JOBBRA gombot a telefonon, akkor küldi ezt a jelet. Ekkor a program alapállapotba forgatja vissza a kanyarodó motort.
- 5: A 8-as vagy a LE gomb lenyomása után telefon ötös értékű bájtot küld a téglának. Ezt a jelet érzékelve a program a motorok sebességét 900 szög/másodpercre állítja és hátrafele kezdi őket forgatni.

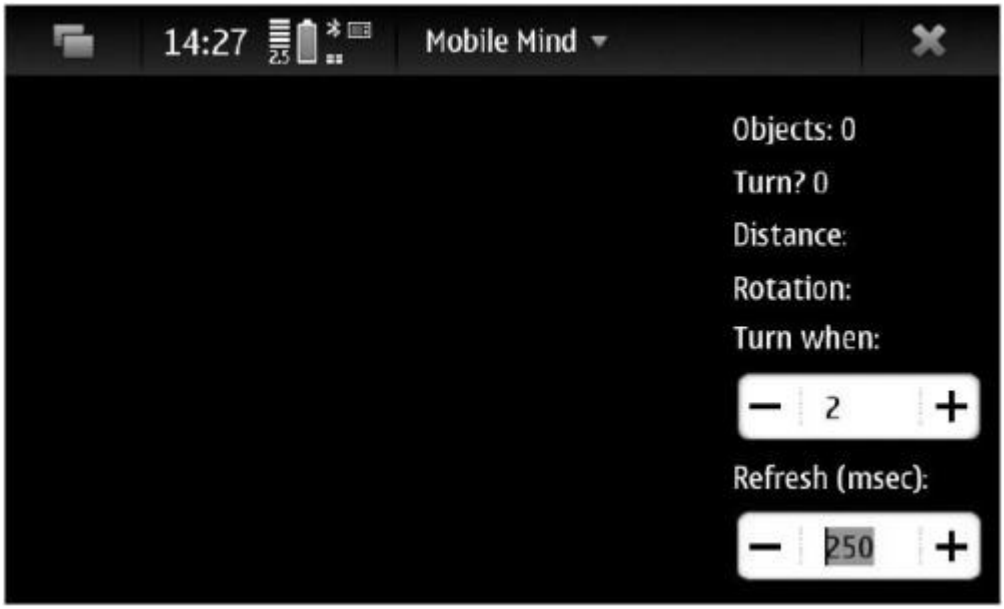
A programszál definiálása után a main metódus elindítja a szálat. Ezután elindít egy ciklust, ami szintén addig fut, amíg a running adattag értéke igaz. A ciklus mindössze egy feltételes utasítást tartalmaz, ami hamisra állítja running értékét, ha lenyomják a vezérlőtégla kilépő gombját.

5.3. Telefon, mint szenzor

Most vizsgáljuk meg azt a projektet, amelyben a mobiltelefon nem, mint felhasználó által kezelt távirányító szerepel, hanem a NXT készlet által szolgáltatott ultrahangszenzor helyettesítője. Itt az érzékelőként használt telefonnak rendelkeznie kell kamerával. Lényegében ez lesz a robotnak a szeme. Az autó az alapján tájékozódik a pályán, amit ezzel a „szemmel” lát. A telefonon felül a pályának is további elvárásoknak kell megfelelnie, hogy a program megfelelően működjön. A pályát határoló külső szalagnak jelen esetben számít a színe is. Fontos, hogy a szalagon egyenlő méretű piros és fehér sávok váltakozzanak, ugyanis a szoftver ezeknek a kameraképen lévő helyzetéből számítja ki, hogy mit kell a robotnak tennie, illetve hol van éppen a pályán. Hogy hogyan, arról részletesebben a program leírásánál fogok beszámolni. Természetesen a távirányítóhoz hasonlóan itt is két programra lesz szükség. Vegyük először a telefonra készült alkalmazást.

5.3.1. MobileMind

Ennek a szoftvernek némiképp komolyabb hardverkövetelményei vannak, mint a szimpla távirányítónak. Ehhez az alkalmazáshoz szükséges egy saját operációs rendszerrel rendelkező úgynevezett okostelefon. A program a készülék kamerája által látott képet dolgozza fel. A feldolgozott adatok alapján küld parancsokat az NXT téglának, hogy a robotnak mit kell tennie. Ezért feltétlenül szükséges, hogy az érzékelő szerepét betöltő telefon képes legyen, minél több képet készíteni és feldolgozni minél rövidebb idő alatt.
Korábban történtek próbálkozások, hogy egy MIDP-s telefonra készüljön egy JavaME-s program, de az így felvett képeknek csupán az elkészítése is több, mint egy másodpercet vett igénybe. Ezért esett a választás olyan telefonra, aminél nem csak JVM-en keresztül, hanem közvetlenül is elérhető a kamerakép.



A program Maemo 5 platformra íródott. A telefon kamerájához Gstreamert használva fér hozzá. Az alkalmazás Nokia N900-as készüléken kb. 150 milliszekundumonként készít és dolgoz fel egy képet. A MobileMind szoftver megfelelő működéséhez a pályának is némiképp speciálisabbnak kell lennie kell lennie, mint a sima Jávácska kupában. Itt fontos, hogy a pályát határoló szalagon egyező színű, azonos méretű, egymástól egyenlő távolságra elhelyezkedő sávoknak legyen felfestve. (Legjobb, ha ugyanolyan széles fehér és színes sávok váltakoznak a szalagon.) A szoftver ezeket a sávokat érzékeli és ezeknek a képen levő helyzetéből számolja ki, hogy a robot merre halad a pályán. Ennek azonban hátránya is van. A pálya környékén nem lehet semmi olyan objektum, aminek színe hasonlít a szalagon lévő sávok színére.

A programot indulása után konfigurálni kell. Először is ki kell választani a kamera által látott képből, a kijelzőt a megfelelő helyen megérintve a szalag sávjainak színét. Be lehet még állítani az érzékenységet, hogy milyen más, de alig eltérő színű pixeleket vegyen a program számításba a sáv meghatározása során. Ki kell választani, hogy hány objektumot kell látnia a telefonnak, hogy kiadja a robotnak a kanyarodási parancsot.
A konfiguráció megadása után csatlakozni kell az NXT téglához. A kapcsolat létrejötte után megkezdődik a kommunikáció. A program a képen először lefuttat egy színszűrőt, mely különálló objektumokként kiadja a sávokat. Ezek után egy Graham szűrővel meghatározza sávok középpontját, ami reprezentálja a sávokat. Ha a felhasználó által beállított értéktől kevesebb sávot lát akkor kiadja a kanyarodási parancsot.

5.3.1. LegoN900

Ez az eredeti robothoz hasonlóan ennek a szoftvere is a Behavior API-t használja fel. Ez a verzió kiegészül két osztállyal. Egy, ami a Bluetoothon keresztül érkező parancsokat dolgozza fel, másik pedig az érkező parancsok által beállított adatokat tartalmazza, amik meghatározzák, hogy a robotnak mit kell tennie. Ki lett dolgozva egy protokoll, ami megmondja, hogy milyen érkező adatra, a programnak mit kell reagálnia.
A fő különbség a korábbi és ezen verzió között, hogy itt a vezérlés során nem közvetlenül a motorokat kezeljük. Itt az irányítást a TachoPilot osztály metódusai végzi. Emiatt a robotnak lényegesen más a felépítése is, mivel a TachoPilot, mint korábban is említettem differenciálműves kormányzást igényel. Ahhoz hogy a TachoPilot metódusai jól működjenek, konstruktorában precízen meg kell adni a kerekek átmérőjét valamint a két kerék középpontjának távolságát valamilyen általánosan elfogadott mértékegységben, de az fontos, hogy a két mértékegység ugyanaz legyen.

5.3.1.1. PhoneData

Ebben az osztályban tárolja a program a telefonról érkező parancsok által meghatározott beállításokat. Tartalmaz egy-egy boolean változót, annak eldöntésére, hogy a robotnak kanyarodnia kell-e (turn), vagy le kell-e állnia (raceOver). Ezen felül egy int változót, ami a robot szemközti faltól mért távolságát tartja nyilván centiméterben (wallDistance). Az

adattagok mind privátak. Az osztály metódusai mindössze az adattagok getter-setter metódusai.

5.3.1.2. BTClient

Ez az osztály felelős a telefonnal való kommunikációért. Az osztály implementálja a Runnable interfészt, azaz tartalmaz egy run() metódust, amelynek utasításai egy külön szálban futnak párhuzamosan a Arbitrator vezérlőjével. Egyetlen adattagja egy NXTConnection objektum.

A run() metóduson kívül egyetlen statikus openBT() nevű metódussal rendelkezik. Ez először létrehoz egy négyelemű bájttömböt, ami a kapcsolat létrehozásához szükséges pin kódot tartalmazza, majd az NXT tégla képernyőjén értesíti a felhasználót, hogy várakozik a kapcsolódásra. Meghívja a Bluetooth osztály waitForConnection() metódusát megfelelően paraméterezve, mely, ha talál kapcsolódásra alkalmas eszközt, visszatér egy BTConnection objektummal, amit az BTClient osztály adattagja kap értékül. Ha a kapcsolat létrejött, a program ezt kiírja a kijelzőre és elindítja az osztály által megvalósított programszálat.

Ebben a programszálban valósul meg a tényleges kommunikáció a telefonnal. Legelőször létrehoz egy InputStream-et az NXTConnection objektumból, valamint három int változót. Az egyikbe a telefonról érkező parancsokat fogja beolvasni (b), a másik kettő segédváltozó, amibe a parancsok paraméterei érkeznek be (p1, p2).

Ezután elindul egy végtelenciklus. Első lépésben kinullázza a változókat, majd beolvassa a parancsot a b változóba. Bár az InputStream read() metódusa int-et ad vissza, de a Bluetooth kapcsolaton keresztül csak byte adatokat lehet küldeni. Ha tehát -1 értéket olvasunk az hibát jelez. Ilyen esetben a program leállítja a ciklus futását.

Ha nem lépett fel semmilyen hiba, akkor a következő lépés megvizsgálni, milyen parancs érkezett be. Nézzük meg, hogy a b változó egyes értékeinél mit csinál a program.

- 0: Egyszerűen beállítja a PhoneData osztály isRaceOver nevű boolean változóját true-ra.
- 1: Ez a parancs a kanyarodásról dönt. Egy paramétere van. Ha a paraméter értéke 1, akkor a PhoneData turn változóját igaz értékre állítja, ha 0, akkor hamisra.
- 2: Ekkor az érkező két paraméterből összerakja a szemközti faltól való távolságot. Ez egy short int érték, de mivel csak bájtokat tudunk küldeni, ezért két részletben

kell a téglának átadni. Először a felső bájt érkezik majd az alsó. Mivel int változókba olvasunk be, megtehetjük, hogy az első változó értékét beszorozzuk 0x100 hexa-számmal és hozzáadhatjuk a másodikat. Ezt az értéket beállítjuk a PhoneData osztály wallDistance változójába.

A ciklus minden lefutásakor kiírja a vezérlőtégla kijelzőjére a parancsot és a paraméterét. Bár ez a felhasználóknak nem jelent sokat, de a fejlesztés során hasznos információ volt.

5.3.1.3. Main

Ez a szoftver fő osztálya. Hasonlóan az szenzoros verzióhoz itt is a sebesség beolvasásával kezdődik a program. Miután ezt a felhasználó megadta, létrehoz egy TachoPilot objektumot az autó pontos adataival. Ezután meghívja a BTClient osztály OpenBT metódusát. Az megpróbál kapcsolatot létrehozni a telefonnal. Ha a kapcsolatfelvétel sikeres volt, ezt jelzi a felhasználó felé és elindítja a BTClient osztály által megvalósított programszálat. A szál elindítása után a program létrehoz egy viselkedéseket tartalmazó tömböt. Ebben létrehozza minden viselkedésosztálynak egy-egy példányát a megfelelő sorrendben. Végül létrehoz egy Arbitrator objektumot, melynek példányosításkor átadja a viselkedéstömböt, és elindítja a vezérlőrendszert.

5.3.1.4. ForwardBehavior

Az előrehaladásért felelős osztály. Az Arbitratornak átadott viselkedéstömb legelső eleme, így ennek a legkisebb a prioritása. Bármelyik másik viselkedés takeControl() metódusa tér vissza igaz értékkel, elveszti az irányítást. Metódusai:

- takeControl(): Minden esetben igaz értékkel tér vissza.
- action(): Meghívja először a Main osztály TachoPilot objektumának setSpeed() metódusát a felhasználó által megadott paraméterrel, majd a forward() metódust.

A Behavior interfész megvalósításaként implementálja a suppress() metódust is, de az nem tartalmaz utasításokat.

5.3.1.5. TurnBehavior

A robot kanyarodását vezérlő osztály. A suppress() itt is egy üres metódus. Az osztály többi metódusa:

- takeControl(): A PhoneData osztály turn adattagjának értékével tér vissza.
- action(): Beállítja a TahcoPilot sebességét 400 szög/másodpercre, majd meghívja a TachoPilot steer() metódusát, 75-ös fordulási aránnyal és 90 fokos fordulási szöggel.

5.3.1.6. StopBehavior

Leállítja a robot motorjait. A suppress() metódus ez esetben is üres. Az osztály többi metódusa:

- takeControl(): A PhoneData osztály isRaceOver adattagjával tér vissza.
- action(): Meghívja a TachoPilot stop() metódusát.

6. Összefoglalás

Az idei évben indult egy projekt a Debreceni Egyetem Informatika Karán Jávácska kupa címmel [3]. Ennek lényege, hogy a kupára versenyistállók, hallgatókból álló csapatok nevezhetnek. A csapatoknak építeniük kell egy robot autót, a Lego Mindstorms NXT készlet felhasználásával. A robotokra az istállóknak a leJOS API segítségével Java nyelven kell szoftvert írniuk és az elkészült kocsik egy szabványos, bárhol könnyedén és gyorsan felállítható pályán versenyeznek egymással.

Az első kutatások, amik a Lego Mindstorms kezdetét jelentették 1987-ben kezdődtek az MIT Média Laboratóriumában. Az első igazi Mindstorms készletet, a Robotics Invention System-et 1998-ban dobták piacra. Ennek a lelke, a még igen szerény képességekkel bíró sárga RCX tégla. Hatalmas sikere ellenére a folytatás még 8 évet váratott magára.

2006. augusztus 2.-án jelent meg végül a Lego Mindstorms NXT. Ez már jóval elegánsabb megjelenésű és komolyabb teljesítményű vezérlőtéglával rendelkezik. A készlet a tégla mellett tartalmaz számtalan Lego építőelemet, három szervo motort és négy szenzor, távolság, érintés, hang és fényszenzort. Már ezekkel az elemekkel is a legváltozatosabb formájú és funkciójú robotokat lehet megalkotni, de még fejlettebbeket lehet építeni a más gyártók által forgalmazott további érzékelők felhasználásával. Az NXT legnagyobb előnye azonban az RCX-szel szemben, hogy az már képes a Bluetooth kommunikációra, amivel a robot képes csatlakozni PC-hez, mobiltelefonhoz, PDA-hoz és más egyéb Bluetooth eszközhöz. Ez szinte korlátlan lehetőségeket nyújt a robotprogramozók számára.

Az NXT robot programozására az egyik lehetőség a Mindstorms készlethez járó NI által kifejlesztett NXT-G. Ez a nyelv kezdők számára ideális, azonban akiknek már van valamennyi programozási tapasztalata érdemes a NXT-G-t leváltani valamelyik magas szintű programozási nyelvre. Egyrészt, mivel könnyebben lehet kezelni egy már jól ismert közkedvelt nyelvet, másrészt egy magas szintű nyelv nyújt olyan szolgáltatásokat is, amelyeket az NXT-G nem. Ilyenek például az objektumorientáltság, lebegőpontos aritmetika, szálkezelés, kivételkezelés, stb. A leJOS NXJ segítségével robotunkra Java nyelven írhatunk programot. Az leJOS biztosít többek közt egy firmware-t a vezérlőtégla gyári szoftvere helyett, linkert, ami a téglán futtatható bájtkódot állít elő a forrásprogramunkból, API-t a robot eszközeinek programozására, PC-s eszközöket a programjaink fejlesztésére és téglára történő feltöltésére.

A motorok kezelésére és adatainak lekérdezésére a lejos.nxt.Motor osztály tartalmaz metódusokat. Ezeket a metódusokat az osztály három statikus példányán (A, B, C) keresztül hívhatjuk meg, melyek az NXT téglán található A, B, és C portokra kötött motorokat kezelik. Némiképp hasonlóan kezelhetőek a szenzorok is. A leJOS-ban minden egyes szenzorfajta – még a nem Lego gyártmányok is – reprezentálására létezik egy-egy osztály. Hogy a szenzorokat használhassuk, rá kell kössük azokat valamelyik szenzor portra. A portokat a lejos.nxt.SensorPort osztálya reprezentálja. Az osztályban a tégla négy szenzorportjának egy-egy statikus példány található (S1, S2, S3, S4). Hogy a szenzor, amit kezelni akarunk tudja, hogy melyik porton zajlik a kommunikáció, ezen a szenzor példány létrehozásakor át kell adnunk ezen statikus példányok közül a megfelelőt a szenzor osztály konstruktorának. Ha sikerült létrehozni a szenzor példányt, akkor ezen keresztül a megfelelő metódushívásokkal lekérdezhetjük az érzékelő mérési eredményeit.

Ha már megy az érzékelők és a motorok programozása, akkor neki lehet kezdeni komplexebb feladatok ellátására képes robotok alkotásának. Ehhez a leJOS Behavior API-ját használhatjuk. Ennek segítségével tudunk írni mesterséges intelligencia programokat olyan kevés memória használattal is, mint amennyivel az NXT is rendelkezik. Az API használatakor viselkedéseket kell programoznunk. A robot általános viselkedését, kisebb elemi részviselkedésekre kell bontanunk és ezeket implementálni egy-egy a Behavior interfészt megvalósító osztályban. A viselkedésosztály megadásakor legfontosabb, hogy egy takeControl() metódusban megadjunk egy ingert (például egy szenzor egy bizonyos értéket mér), ami ha bekövetkezik, akkor igaz értékkel tér vissza a metódus, valamint egy action() metódusban implementáljuk az erre az ingerre adott reakciót. Ha megvannak az osztályaink, meg kell adni azt is, hogy melyik viselkedésnek nagyobb a prioritása, arra az esetre, ha robotot egyszerre két olyan inger is éri, amire reagálnia kell.

A leJOS lehetőséget biztosít arra is, hogy a robotunkkal Bluetooth kommunikációt tudjunk megvalósítani. A kommunikáció során első lépésben meg kell keresni a tégla közelében levő aktív Bluetooth készülékeket a getKnownDevices() metódussal. Ez a hívás csak azokat az eszközöket adja vissza, amihez a tégla korábban már csatlakozott és a kapcsolathoz szükséges adatok a memóriájában tárolva vannak. Ha olyan készüléket akarunk, megtalálni, amivel a tégla még nem találkozott korábban az inqiure() metódust kell használni. Ha megtalált készülékek között van, amelyikkel a kommunikációnak a programban folynia kell, akkor

csatlakozni kell hozzá. Sikeres csatlakozás esetén a kapcsolat objektumból létrehozott Input és OutputStreamek író/olvasó metódusaival megkezdődhet a kommunikáció.

A Jávácska kupa robotjai csak a Behavior API-t használják, a Kiterjesztett Jávácska kupa azonban már a Bluetooth kommunikációt is megvalósít. Az alap kupán csak a meglehetősen pontatlan ultrahang szenzort lehet használni, ezért ezen a téren még nem történt átütő siker. Egyelőre az igazi versenyt még nem sikerült elindítani, mivel a szenzor és a motorok pontatlanságai miatt a robotok egy kört is nehezen tesznek meg a pályán. Jelenleg csak időmérő verseny van. A leggyorsabb köröket futó autót építő istálló a nyertes. Folynak kutatások egy olyan megoldás kialakítására, melyben a pályát egy bittérkép reprezentálja. A mátrixnak azon eleme lenne igaz, ahol a robot éppen jár. Ezt a motorok állását mutató adatok lekérdezéséből lehetne kiszámolni. Ekkor a távolságszenzornak csak a pályán haladó ellenfelekre kellene figyelnie.

A kiterjesztett kupában jó ötletnek bizonyult a szenzor lecserélése egy okos telefonra. Amikor a robotra egy Nokia N900-as telefon volt erősítve és a telefon a kameraképének feldolgozásából nyert adatok alapján tájékozódott, akár hat kört is képes volt megtenni a pályán. Természetesen egy idő után ez a robot is eltért a megfelelő iránytól. Ebből a robotból is készül a fejlesztett verzió, amelyik nemcsak a pálya szemközti, hanem az oldalfalát is figyelve kiszámítja, hogy mennyire tér el a robot az egyenestől, és korrigálja azt.

A Kiterjesztett Jávácska kupa koncepciója és megvalósítása annyira sikeres volt, hogy a Nokia Calling All Innovators versenyen különdíjat nyert [13].

7. Köszönetnyilvánítás

Szeretnék köszönetet mondani Bátfai Norbert tanár úrnak, hogy lehetővé tette a tanszéki Lego Mindstorms készletek használatát, létrejöhetett a Jávácska kupa, valamint a diplomamunka megírásában nyújtott segítségéért.

Ezen felül külön köszönetet mondanék Balázs Ádámnak és Kovács Zsoltnak, akikkel egy csapatban dolgoztam az alap és a kiterjesztett Jávácska Kupa robotjain és a szoftverein.

8. Irodalomjegyzék

1. Bátfai N., Molnár P., Molnárné Nagy M., Rábai B., Szitha K., Kovács Zs., Hudák L., Rák J. (2010): A Debreceni Fejlesztői Hálózat, Híradástechnikai Szemle LXV: 5-6.
2. Norbert Bátfai, László Hudák, Dávid Doszpoly, Károly Frendrich, János Arpád Orbán, János Rák: Jávácska Cup: Developing a Standard Robot Car Racing Platform (kéziratban 2010)
3. Bátfai N., "Mobiltelefonos játékok tervezése és fejlesztése". PhD doktori disszertáció. 2010. http://www.inf.unideb.hu/~nbatfai/phd, http://www.inf.unideb.hu/~nbatfai/phd/MobiltelefonosJatekokTervezeseEsFejlesztese Ertekezes.pdf 125. o.
4. http://www.inf.unideb.hu/~nbatfai/JegymegajanloDolgozatValaszthatoFeladatai.pdf 6-8. o.
5. Consturctopedia http://www.education.rec.ri.cmu.edu/content/lego/building/media/Constructopedia%202.pdf
6. Brian Bagnall: Maximum Lego NXT: Building Robots with Java Brains, Variant Press 2007
7. leJOS, Java for Lego Mindstorms, http://lejos.sourceforge.net/
8. Juan Antonio Brena Moral: Bluetooth and leJOS, http://www.juanantonio.info/p_articles/archive/2008/LEJOS-BLUETOOTH.pdf
9. R. A. Brooks. Intelligence without representation. Artificial Intelligence, http://people.csail.mit.edu/brooks/papers/representation.pdf
10. M. W. Lew, T. B. Horton, and M. S. Sherri . Using lego mindstorms nxt and lejos in an advanced software engineering course http://www.cs.virginia.edu/~sherriff/papers/CSEET2010-Lew.pdf
11. LEGO.com MINDSTORMS http://mindstorms.lego.com/
12. Lego Mindstoms – Wikipedia, http://en.wikipedia.org/wiki/Lego_Mindstorms
13. https://www.vik.bme.hu/aktualis/hirek/218/