

TDK dolgozat

Dóczi Roland

# Fürkész, az autonóm robotautó

**Témavezető:** Dr. Végh János  
Dsc, egyetemi tanár

**Készítette:** Dóczi Roland  
mérnök informatikus, BSc

2010/2011/1

Debreceni Egyetem Informatikai Kar  
Informatikai Rendszerek és Hálózatok Tanszék

Debrecen

2014

## Tartalom

|       |                                                               |    |
|-------|---------------------------------------------------------------|----|
| 1.    | BEVEZETÉS.....                                                | 5  |
| 2.    | CÉLKITŰZÉSEK.....                                             | 5  |
| 3.    | ROBOTOKRÓL/ROBOTIKÁRÓL ÁLTALÁNOSAN .....                      | 6  |
| 3.1   | FOGALMA .....                                                 | 6  |
| 3.2   | AZ ELSŐ GENERÁCIÓS ROBOTOK.....                               | 6  |
| 3.3   | A MÁSODIK GENERÁCIÓS ROBOTOK .....                            | 7  |
| 3.4   | A HARMADIK GENERÁCIÓS ROBOTOK .....                           | 7  |
| 3.5   | TÖRTÉNELMI ÖSSZEFOGLALÓ .....                                 | 7  |
| 3.6   | AUTONÓMRENDSZEREK A NAGYVILÁGBAN .....                        | 8  |
| 4.    | ROBOTOK FELÉPÍTÉSE .....                                      | 9  |
| 4.1   | ÉRZÉKELÉS .....                                               | 9  |
| 4.2   | LÁTÁS .....                                                   | 10 |
| 4.3   | ELMOZDULÁS .....                                              | 10 |
| 4.4   | KÜLSŐ KÖRNYEZETI TÉNYEZŐK MÉRÉSE .....                        | 10 |
| 5.    | AZ ESZKÖZ RÖVID BEMUTATÁSA .....                              | 11 |
| 6.    | A ROBOTON HASZNÁLT MODULOKRÓL .....                           | 12 |
| 6.1   | A MIKROKONTROLLEREKRŐL ÁLTALÁNOSÁGBAN .....                   | 12 |
| 6.2   | AZ ÁLTALAM HASZNÁLT MIKROKONTROLLERRŐL ÉS PERIFÉRIÁIRÓL ..... | 14 |
| 6.3   | AZ ÁLTALAM HASZNÁLT HARDVERELEMELK.....                       | 16 |
| 6.3.1 | VÁZ .....                                                     | 16 |
| 6.3.2 | MOTOR .....                                                   | 16 |
| 6.3.3 | ULTRAHANGSZENZOR.....                                         | 17 |
| 6.3.4 | GIROSKÓP/GYORSULÁSMÉRŐ .....                                  | 18 |
| 6.3.5 | BLUETOOTH.....                                                | 19 |
| 6.4   | A FEJLESZTŐI KÖRNYEZET ÉS A FELHASZNÁLT PROGRAMCSOMAG .....   | 20 |
| 7.    | AZ ANDROID, MINT FŐFEDÉLZETI RENDSZER.....                    | 22 |
| 7.1   | AZ ANDROID FEJLŐDÉSE .....                                    | 22 |
| 7.2   | AZ ANDROID PROJEKT BEN HASZNÁLT KÜLSŐ PROGRAMCSOMAG .....     | 24 |
| 7.3   | AZ ÉRZÉKELŐK MŰKÖDTETÉSE.....                                 | 26 |
| 7.4   | A PROJEKT FUNKCIONÁLIS ELEMEI RÉSZLETESEN .....               | 29 |
| 8.    | A MEGVALÓSÍTÁS.....                                           | 30 |
| 8.1   | A KEZDETEK .....                                              | 30 |

|                                   |    |
|-----------------------------------|----|
| 8.2 A TÉNYLEGES MEGVALÓSÍTÁS..... | 30 |
| 9. ÖSSZEFOGLALÁS .....            | 38 |
| 10. IRODALOMJEGYZÉK .....         | 39 |
| 11. KÖSZÖNETNYILVÁNÍTÁS .....     | 43 |

## **1. BEVEZETÉS**

Egyetemi tanulmányaim alatt sikerült elmélyülnöm az elektronika, illetve az objektumorientált programnyelvek világában. Ez a két téma felkeltette az érdeklődésemet, ezért arra gondoltam, hogy egy ezen a területen kezdek tevékenykedni. Dolgozatom keretében úgy gondoltam, hogy egy robotautót építek, ami képes önállóan mozogni, felszerelve egy kezdetleges mesterséges intelligenciával. Ebben főként a Vasember [1] (Iron Man) filmek is motiváltak, és tereltek tovább az előbbi kettőt ötvöző autonóm rendszerek világába. Ugyanis ebben a trilógiában a főszereplő egy saját maga által létrehozott, majd mesterséges intelligenciával felszerelt és önálló működésre képes páncélt, illetve egyéb eszközöket hozott létre. Ez a filmtrilógia motivált a legjobban a dolgozatom elkészítése közben. Ezek után egy olyan autonóm robotautó megépítése volt a fő célom, mely képes eljutni olyan helyekre is, amelyek nem biztonságosak. Ilyenek helyek lehetnek katasztrófa sújtotta területek, romok, vagy akár olyan helyek is, ahol valamilyen mérges vegyszer van a levegőben. Ezen felül még, hogy ha egyszer bejuttatták egy ilyen helyre, onnan önállóan legyen képes felfedezni.

## **2. CÉLKITŰZÉSEK**

Dolgozatom fő célkitűzése egy olyan autonóm robotautó építése volt, mely képes önálló mozgásra, az akadályok elkerülésével. Működése egy 4, vagy több ultrahangszennorral megvalósított sztochasztikus folyamatnak mondható, mivel az volt az elvárás, hogy az autó önállóan járja be a környezetét. Ezeket a célokat meglepően gyorsan sikerült abszolválnom. Ezek után egy nagy döntés állt előttem a fejlesztésben. Egyik egy egyszerű távirányítós autó megvalósítása volt, míg a másik egy önállóan működő robotautó. Az előbbi kitűzés egyszerűbbnek tűnhet, viszont ha az utóbbit választom, sokat tanulhatok belőle. A felfedezés itt azt jelenti, hogy szolgáltatasson adatokat a környezetéről minél részletesebben, pontosan, és valós időben. Ilyen adatokat a GPS, kamera, giroszkóp, gyorsulásmérő, iránytű, hőmérő, és ultrahang modulok szolgáltathatnak. Fontos a szinkronizáció, illetve a modulok terheltségének elosztása, megfelelő vezérlés kiépítése. Ezek megvalósítása mellett egy biztosan működő rendszert kaphatunk eredményül. A végső cél az előbb említett eszköz egy olyan prototípusának a létrehozása, ami a tényleges megvalósításához tökéletes modellt alkot.

### 3. ROBOTOKRÓL/ROBOTIKÁRÓL ÁLTALÁNOSAN

#### 3.1 FOGALMA

Magát a robot szót Karel Čapek, 20. századi cseh író használta először R.U.R. [2] (Rossum's Artificial Robots) című színdarabjában. Ma, amikor a robot szót halljuk, valószínűleg nem a Čapek által elképzelt és megformált, egy élő emberre megtévesztésig hasonlító, önállóan gondolkodó és a munkát látszólag boldogan végző, szintetikus organikus anyagokból létrehozott lakótárs jut eszünkbe először. A darab nagy sikert aratott, premiere után rögtön beégette a robot szót először az angol nyelvbe, majd a science fiction világába. Čapek nem definiálta a szó. Cseh nyelven a robot *robota*, ami magyarra lefordítva munkát jelent.

A szó első definiálására a DVI 2860-as szabványban [3] került sor, 1990-ben. A szabvány szerint: *Automatikusan irányított, újraprogramozható, többcélú, manipulátor programozható három, vagy több tengelyen, ami vagy egy helyhez rögzített, vagy mobilizált különböző ipari tevékenységek automatikus elvégzésére.*

Az 1940-es évek elejéig azok a személyek, akik robotépítésbe kezdtek, nem tudták pontosan, mi is számít annak. 1942-ben viszont Isaac Asimov, orosz származású író, biokémikus az *Én a robot* című novellájában először írta le a robotika három törvényét [4]:

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

#### 3.2 AZ ELSŐ GENERÁCIÓS ROBOTOK

Az 1950-es években Joseph Engelberger és George Devol alkotta meg az első ipari robotot, ami valóban hasznos munkát tudott végezni, az Unimate nevet kapta, ami nyers magyar fordításban univerzális társat, vagy egyedi társat jelent. 1954-ben Devol szabadalmaztatta a robottechnológia szabványait. Szélesebb körökben való terjesztésüket segítve Engelberger

1956-ban megalapította a Unimation Inc.-t, és ezzel megindult a robotika terjedése. Először a General Motors gyárába telepítettek egy ilyen gépet, 1961-ben [5].

Az Engelberger-féle robot fizikailag rögzítve volt. Az első mozgásra képes robotot 1960-ban a Johns Hopkins egyetem alkalmazott fizika tanszékén fejlesztették ki, és a Johns Hopkins Beast [6] nevet viselte. Belső programja és ez által az intelligenciája még kezdetleges volt. Belső programja nem volt, nem CPU vezérelte a működését. Rengeteg tranzisztorból álló áramkör működtette. Működése annyiból állt, hogy a homogén fehér falú laboratóriumban képes volt megkeresni a fekete aljzatokat. Ha talált egyet csatlakozott rá, majd várt, amíg feltöltődik.

### **3.3 A MÁSODIK GENERÁCIÓS ROBOTOK**

Az 1972-es évre az SRI International fejlesztette ki az első általános célú robotot, Shakey-t. Ez a robot már lépes volt egy berendezett szobáról, egy másik szobára vándorolni, ez alatt akadályokat kerülni és a villanykapcsolókat használni. Esetleg ha szükséges, a tárgyakat arrébb tudta tolni. Véleményem szerint kezdetleges mesterséges intelligenciájához képest, ez az eszköz már hasonlít a mai tudatba vésődött robot fogalmához.

### **3.4 A HARMADIK GENERÁCIÓS ROBOTOK**

Az ebbe a kategóriába tartozó robotok képesek az önálló döntéshozatalra, mesterséges intelligenciával rendelkeznek. Képesek megtanulni az egyes tárgyak alakját, majd felismerni azokat. Képesek a tanulásra. Általában automatizált szerelés ellenőrzésére alkalmazzák, de egyéb komplex feladatok ellátására is képes.

### **3.5 TÖRTÉNELMI ÖSSZEFOGLALÓ**

Ahogy az előző felsorolásból egyértelművé vált, kezdetben a robotok fejlesztésének célja nem a mai akció és science-fiction filmekben bemutatott filmes klisé (miszerint például robotokkal tökéletesen le igázható egy egész faj) volt, hanem az emberiség életének könnyebbé tétele. Asimov magát azt a tényt, miszerint a robotnak nem szabad kárt tennie az emberiségben, hamar felismerte, ezért kibővítette a három alaptörvényt egy 0.-kal is:

*A robotnak nem szabad kárt okoznia az emberiségben, vagy tétlenül tűnnie, hogy az emberiség bármilyen kárt szenvedjen.*

A 0. sorszámot választotta, mert fontosnak tartotta, hogy a felsorolás fontossági sorrendje megmaradjon.

### **3.6 AUTONÓMRENDSZEREK A NAGYVILÁGBAN**

Ha kimondjuk azt a szókapcsolatot, hogy „autonóm robotok”, valószínűleg a legtöbb embernek a múlt hónapokban nagy hírességnek örvendő Curiosity Mars járó robot jut az eszébe, vagy esetleg a science-fiction filmekből ismert óriási robotok. Utóbbival a dolgozatomban sajnos nem volt idő foglalkozni. A Curiosity [7] robot a NASA egy projektjén belül valósult meg azzal a célkitűzéssel, hogy a Marson található Gale krátert felfedezze. 2011. november 26-án indult útjának az Egyesült Államok Légierőjének bázisáról, Cape Canaveralról. 2012 augusztus 6-án sikeresen landolt is a kráterben, és azóta is szolgáltatja a mérési adatokat a Földre. 2012 decemberében a NASA nagy bejelentette – nagy beharangozó kampány mellett –, hogy egy történelmi felfedezést tettek. Azonban ez a felfedezés mégsem bizonyult akkorának, mert nem sikerült 100%-ig bizonyítani. Elmondásuk szerint  $^{13}\text{C}/^{12}\text{C}$  szénizotóp-arányt mértek, amit azonban nem sikerült bizonyítani, hogy a robot műszerein volt csak ez egyfajta szennyeződés formájában, vagy valóban ott volt a Mars felszínén. Több szakértő szerint, mint Sik András, aki az ELTE Természetföldrajzi Tanszék munkatársa, a talált molekulák nem minden összetevője marsi eredetű [8]. Ezután a kicsit félretájékoztató bejelentés óta óvatosabbak a NASA emberei, eget rengető bejelentéssel még nem érkeztek. Érdekes a robottal kapcsolatban, hogy teljesen autonóm módon kell működnie, mivel a rászerezett kamera képének majdnem 600 millió kilométert kell megtennie, ami az átvittet meglehetősen lelassítja. A robotot úgy építették meg, hogy az összegyűjtött információt először egy átjátszó állomásra küldi, ami a nagyobb teljesítményű antennái segítségével közvetlenül a Földre képes küldeni az adatokat nagyobb sebességgel, mint amekkorával a Curiosity küldené.

Az autonóm rendszerek mai világának másik főszereplője a Google egyik projektje, amelyben egy személyautót alakítottak át sofőr nélküli járművé [9]. A projekt vezetője szerint az általuk fejlesztett algoritmussal és rendszerekkel felvértezett személyautó biztonságosabban vezet, mint egy ember. Ezt nem csak belekiabálták a nagyvilágba. Ugyanis addig lobbiztak az

Egyesült Államokban, amíg Nevadában 2011. június 29-n, és Michigan államban 2013 decemberében legalizálták az automatizált autók, tesztelését közutakon. 2012 augusztusában a Google bejelentette, hogy 500000 km-t vezetett le az autójuk addig, és azt mind balesetmentesen. Az egyetlen baleset, aminek résztvevője egy Google kocsis volt, az emberi figyelmetlenségből adódott, ugyanis Jakartában nekiment egy kisbusznak. Érdekességük ezeknek a járműveknek, hogy a hátsó ülésükön egy komplett szerverpark található, mivel nagymennyiségű adatot kell feldolgoznia nagysebességgel, mert nem mindegy, hogy 0.2 másodperc múlva „tapos” a fékpedálra, vagy 2 másodperc elteltével.

Nem csak a földön, a levegőben is egyre nagyobb szerepet kapnak az autonóm irányított eszközök. Ilyenek a pilótánélküli drónok. Ezeknek nem mai találmányok, már az 1800-as évek közepén is szerepet kaptak az elődeik. Ekkor ugyanis az osztrákok egy „pilótánélküli” bombával töltött léghajót küldtek Velence irányába. Ez azonban elég kezdetleges modell, és nem is hasonlít a mai használatos drónokra. A ma ismert ilyen pilótánélküli eszközök fejlesztése már az 1900-as évek elején elkezdődött, eredeti céljuk az volt, hogy a katonák gyakorolják a célba lövést. Mára nagyot fejlődtek. Körülbelül 50 ország használja a drónokat eltűnt emberek megkeresésére mind a nyíltvízen, mind a szárazföldön, romok közé szorulva.

## **4. ROBOTOK FELÉPÍTÉSE**

Egy robot autonómiáját véleményem szerint legcélszerűbb az emberi szervezet, érzékelő központok „működésével” hasonlóan megtervezni, majd megvalósítani. Az emberi szervezet, mint az idegrendszer, vagy az agy, a legösszetettebb dolgok közé sorolható, mindenre képes reagálni, mind pozitív, vagy negatív reakcióval.

Kutatásaim eredménye szerint ezen idea által vezérelve kell elkezdni egy robot fejlesztését.

### **4.1 ÉRZÉKELES**

Egy robotot, - mivoltából adódóan - egy ágensnek tekinthetjük, melynek környezete a körülötte lévő fizikai tér, a valós világ. Az adott helyzetéből adódó, rá ható ingerekre, külső hatásokra egy viselkedéssel reagál. Szenzorjai és beavatkozási a különböző kölcsönhatások figyelésén kívül saját viselkedésének sikerességét is figyelni tudja.

## 4.2 LÁTÁS

A látás egy olyan élő szervezet számára, mint az ember, az egyik legfontosabb, és egyben legnélkülözhetetlenebb dolog. Egészséges szemekkel rendelkező ember elképzelni sem tudja, milyen problémákkal kell megküzdeniük a látássérülteknek. Egy robotnak, főleg, ha nagyméretű, elengedhetetlen a tökéletes „látás”, különben kárt tenne valamiben, vagy valakiben (robotika 1. főtörvénye ezt nem engedi meg). Ezt különböző módon lehet megvalósítani. Két nagy csoportba sorolnám a látást megvalósító eszközöket, módszereket:

- Kamera/Kamerák képeinek elemzésével látja a környezetét
- Különböző érzékelők segítségével tárgyak helyzetét, illetve relatív távolságával képes vizsgálni a körülötte lévő világban

## 4.3 ELMOZDULÁS

Az elmozdulás vizsgálására több lehetőségünk van, mint a fent említett látásra.

- Giroszkóp: Ez a modul szögelfordulásra reagál.
- Gyorsulásmérő: Dőlést tökéletesen lehet vele vizsgálni a három főtengely irányába.
- GPS: Konkrét hosszúsági és szélességi fokok alapján meghatározható az eszköz helyzete, majd a haladás során megtett távolság vektora alapján meghatározható az elmozdulás iránya, sebessége.
- Magnetométer: A valós északtól való eltérést mutatja. Ha a robotunk saját tengelye körül végez forgómozgást, meghatározható az északtól való eltérés fokokban.

## 4.4 KÜLSŐ KÖRNYEZETI TÉNYEZŐK MÉRÉSE

Külső környezeti tényezők lehetnek például hőmérséklet, a légkörben előforduló kémiai anyagok mennyisége, szélerősség, fényerő. Az érzékelőkre a hőmérséklet ingadozása van a legnagyobb hatással, az érzékelőkben lévő ellenállásoknak 50-60 °C-os hőmérsékletingadozás esetén megváltozik a karakterisztikája. Ezért fizikai mennyiség mérésére alkalmas eszközbe általában allokálnak egy hőmérsékletmérésre alkalmas modult is, hogy az előbb említett nagyobb ingadozások mellett is tudjunk mérni pontos, korrigált értékeket.

## 5. AZ ESZKÖZ RÖVID BEMUTATÁSA

Dolgozatom egy autonóm robotautó megvalósításáról és annak menetéről szól. Működését különböző modulok együttes működése határozza meg. Két vezérlőegység működteti.

Az egyik egy Texas Instruments által forgalmazott Stellaris LM4F120H5QR típusú LaunchPad [10]. Ez a mikrokontroller végzi a motorok vezérlését, a giroszkóp/gyorsulásmérő adatainak begyűjtését, majd ezek továbbítását. Ez Bluetooth-on keresztül történik. Ugyanezen protokollon keresztül fogad is adatokat a másik vezérlőegységtől. Ez kétfajta lehet:

- GPS koordinátákból számított szükséges elfordulás nagysága ahhoz, hogy a cél irányába álljon az autó, illetve, hogy melyik égtáj felé van a cél (Észak-kelet, Dél-kelet, Észak-nyugat, Dél-nyugat).
- Az északhoz viszonyított elfordulás

A Stellaris mikrokontrolleren futó program a fogadott adatok elemzése után eldönti, hogy melyik irányba kell forgatni a kocsit, hogy minél hamarabb irányba álljon, illetve ha akadály van előtte, merre kerülje ki azt. Amennyiben lehetséges, kikerülés közben is a cél irányába halad.

A másik vezérlőegység a fedélzeti rendszer, ami egy okostelefon, Android [11] operációsrendszerrel felvértezve. Ennek feladata a saját moduljainak működtetése, adatainak begyűjtése, továbbítása. Az operációsrendszer egy beépített szolgáltatását használva egy WiFi-n keresztüli mobil hozzáférési pontot hoz létre. Ennek segítségével egy hálózati kiszolgálót futtat, ahol a felhasználó megtekintheti a giroszkóp, gyorsulásmérő valós időben leolvasott értékeit grafikonon ábrázolva, a telefon kamerájának a képét 2 másodpercenkénti frissítéssel és az aktuális GPS koordinátákat. Továbbá a felhasználó beavatkozhat az eszköz működésébe is. Választásom azért esett egy okostelefonra a modulok egyenkénti beszerzése helyett, mert egy ilyen készülék ma már megtalálható az emberek nagy többségénél, és magában foglalja az összes modult, amire szükségem van a fejlesztés során. Így a költséghatékonyság jobb lesz.

## 6. A ROBOTON HASZNÁLT MODULOKRÓL

### 6.1 A MIKROKONTROLLEREKRŐL ÁLTALÁNOSSÁGBAN

Mi is az a mikrokontroller? A mikrokontroller [12] egy alacsony energiaigényű kis számítógép egyetlen integrált áramkörön, ami tartalmazza magát a processzort, a memóriát (RAM), kiegészítve a megrendelő által igényelt I/O perifériákkal. Az mindennapi élet számos területén találkozhatunk olyan eszközökkel, melyek mikrokontrollert használnak. A számítógépek, telefonok, televíziók, órák, autók, sőt már az intelligens hűtők, illetve mosógépek is használják ezeket. Főként olyan helyeken találkozhatunk velük, ahol valamilyen automatizálási feladatot kell ellátni, vagy kisebb számítási feladatot rövid időn belül kell elvégezni. A tervezők arra törekcszenek, hogy az eszköz fogyasztását minél kisebbre redukálják, minél több periféria beépítése mellett.

A mai mikrokontrollerek átlagos áramfelvétele 100mA körülire tehető, 1.8V, 3.3V, vagy 5V tápfeszültség mellett, a processzor sebességét általában 1MHz és 120MHz közé. Programozható memóriája FLASH, általában 100 byte és 100 kByte közti tárhellyel. RAM memóriája néhány byte-tól 300-400 byte-ig. Továbbá rendelkeznek 4 – 128 GPIO lábbal.

Tokozása négy fajta lehet (tokozás alatt a külső burok, és érintkezők kialakítását értjük):

- SIL (egyoldali)
  - 1-soros
  - 2-soros
- DIL (kétoldali)
  - 2-soros
  - 4-soros
- QIL (négy oldali)
- GA (Grid Array, alsó mátrix)

Fontos jellemzője még a szóhossza (belső adatméret), ami lehet 8, 16, és 32 bites.

Az első mikrokontroller az Intel cég gyártotta le 1971-ben, ami az Intel 4004 volt. Ez egy 4 ites 740 kHz sebességű 16P DIP tokozású integrált áramkör volt, 4 bites BCD (Binary-coded decimal) utasításkészlettel. Számológépekben használták egyszerű számolások végrehajtására, illetve aritmetikai műveletek elvégzésére.

Egy mikrokontroller GPIO (General Purpose Input Output) lábai sokrétű feladatokat láthatnak el. Alapesetben nincs speciális mód definiálva külön-külön az összesre, a felhasználó által írt program utasításai alapján kerülnek kiosztásra. Vannak statikus lábak is, aminek a bemenetét nem lehet módosítani, mint például a VCC (tápfeszültség), VDD (föld), vagy az RST (reset). A mikrokontroller a programot a FLASH memóriában tárolja. A FLASH mivoltából adódóan ez akár több ezerszer is újraprogramozható. A vezérlő lábait két nagy csoportra lehet osztani, ezek lehetnek digitális I/O, vagy analóg I/O. Bemeneti lábakat arra használjuk, hogy különböző mérési eredményeket, vagy esetenként megszakításokat, érzékelők válaszait tudjunk fogadni, kezelni, reagálni azokra. Kimeneti lábakkal lehetőségünk van például kapcsolókat használni, LED-eket működtetni, LCD kijelzőre írni, modulokat triggerelni, vagy kommunikálni különböző protokolloknak megfelelően. A legegyszerűbb felhasználása egy I/O lábnak az, ha digitális kimenetként használjuk. Ilyenkor magas, vagy alacsony feszültség szint között választhatunk, melyiket adjuk ki a szóban forgó lábra, így a lábon át egy másik eszközre.

## 6.2 AZ ÁLTALAM HASZNÁLT MIKROKONTROLLERRŐL ÉS PERIFÉRIÁIRÓL

Munkám során egy Texas Instruments által forgalmazott Stellaris LM4F120H5QR LaunchPadot használtam.



I. ábra, Stellaris LaunchPad

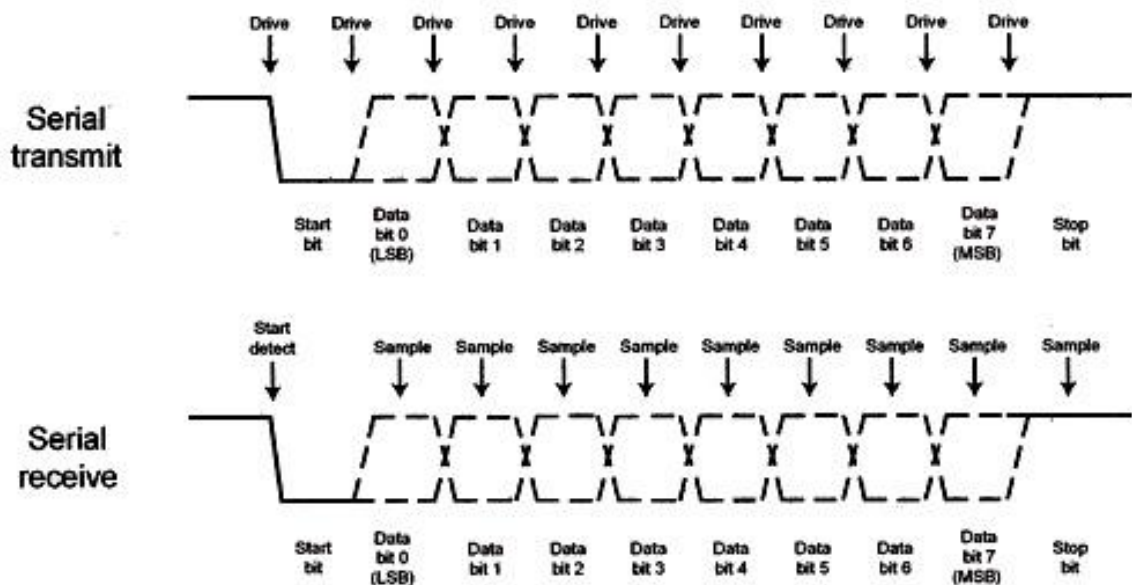
Specifikációja:

- Tokozás: 64LQFP
- CPU: ARM Cortex-M4Flash: 258 (KB)
- SRAM: 32 kB
- Maximális órajel: 80 MHz
- GPIO: 43 db
- USB D
- SSI/SPI: 4 db
- I2C: 4 db
- UART: 8 db
- ADC: 12 db
- ADC felbontása: 12
- CAN MAC: 1 db

- Működési hőmérséklete:  $-40 - 85\text{ }^{\circ}\text{C}$
- Timer: 6 db 64 bites és 6 db 32 bites

Egy mikrokontrollerben a CPU, a FLASH, az ALU és a RAM mellett – ahogy fent is látható – számos periféria is helyet foglalhat. Ezek közül itt csak azokat emelném ki, amelyeket én is használtam a munkám során.

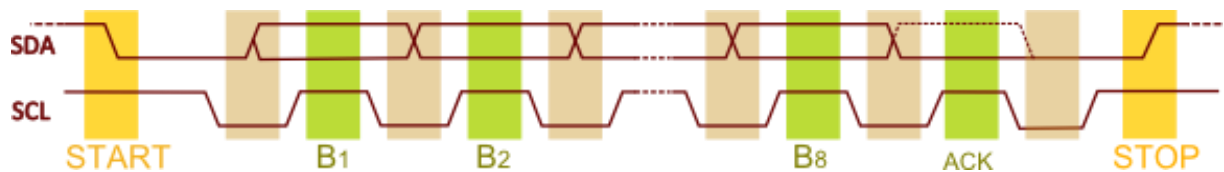
*UART (Universal Asynchronous Receiver/Transmitter)* [13]: Magyarra fordítva Univerzális Aszinkron Adó/Vevő. Az univerzális szó a konfigurálhatóságára utal. Ez azt jelenti, hogy módosítható az átviteli sebesség, és az adat formátuma is. Átvitel esetén start, illetve stop kódokkal határolt bitfolyammá alakítja át a küldendő adatot. Egy átvitel így a start bit alacsony szintje jelzi az átvite kezdetét, majd 8 jelváltásnyi idő után a stop bit magas szintje jelzi a végét. Az eleje és a vége közti 8 jelváltásnyi idő az átvitt adat egy byte-át reprezentálja. A jelváltás (baud) azt jelenti, hogy egy másodperc alatt hányszor képes az UART modul a küldést végző lábat magasról alacsony, vagy alacsonyról magas szintre emelni.



**II. ábra, Az UART működése**

*I2C* [14]: Modulok közti kétvezetékes szinkron adatátvitelt tesz lehetővé 100 kHz, 400 kHz, vagy 1 MHz sebességgel. A Philips cég találta fel, integrált áramkörök összekapcsolására. Egy I2C modulon a VCC, illetve a GND lábak mellett két adatláb is helyet kap. Ezek az órajel (SCL), illetve az adat (SDA) lábak. Működés közben alapvetően egy master, és emellett egy, vagy több slave eszköz kommunikál egymással. A master-slave kapcsolat lényege az, hogy a

kommunikáció során mindig a masternek választott eszköz kezdeményezi és vezérli azt. A slave eszközök mindegyike rendelkezik egy egyedi címmel, ami a közös busz használatakor elengedhetetlen. Speciális esetekben több master is jelen lehet, ezt multi-masternek nevezzük. Itt az UART-tól eltérően nem egy start bit alacsonyszintre emelésével kezdődik az adatátvitel. Kommunikáció akkor indítható, ha a busz nem foglalt. A szabad állapot jellemzője, hogy mind az SDA és mind az SCL lábak magas szinten vannak egy STOP feltételt követően. Az átvitel 8 biten történik.



**III. ábra, Az I2C feszültségszintjei**

Bekötés során az SDA és SCL lábakat felhúzó ellenállásokkal kell biztosítani. Erre azért van szükség, mert az I2C vonalain mindkét irányban áramolhatnak jelek, és ezek szembekapcsolásának megakadályozására minden kimenet nyelőelektrodás, melyek csak lefelé húznak. A felfelé húzást felhúzó ellenállásokkal valósíthatjuk meg.

## **6.3 AZ ÁLTALAM HASZNÁLT HARDVERELEMELMEK**

### **6.3.1 VÁZ**

Az autó vázát nem saját kezűleg építettem meg, alkatrészei megvásároltam.

Specifikációja:

- 14cm x 25cm nagyságú váz
- Anyaga átlátszó, műanyag

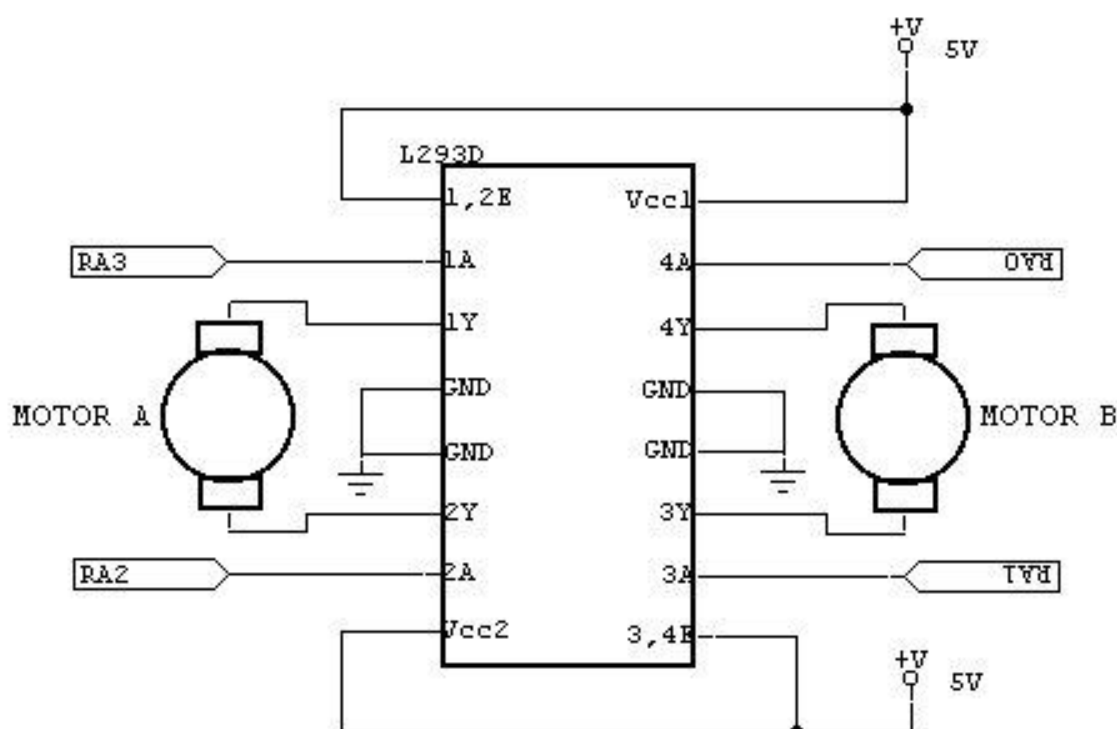
### **6.3.2 MOTOR**

Az autó vázához rögzített négy kerék mindegyikéhez tartozik 1-1 dedikált motor. Ezek DC [15] motorok, ami azt jelenti, hogy egyenárammal kell táplálni őket. Az ilyen típusokban az áram, a fordulatszám, a feszültség és a nyomaték egyenesen arányos.

Az általam használt DC motor pontosabb specifikációja:

- Üzemeltetési feszültség: maximum 6V
- Teljesítmény: ~200 fordulat másodpercenként
- Áramfelvétel: 120mA
- 240 fordulat/másodperc kerekkel súlyozva, 6V-os táplálás mellett
- 48 méter percenként, 6V-os táplálás mellett

A motorok vezérlését egy H-híd [16] IC-vel a legegyszerűbb megoldani. Ilyen az ST L293D típusú IC-je.



IV. ábra, Az L293D modul helyes bekötése

### 6.3.3 ULTRAHANGSZENZOR

Az ultrahangszenzor egy HC-SR04 [17] típusú modul. üzemeltetési feszültsége 5V. 2cm és 400cm közti távolságot képes mérni. Működése az operációsrendszerekből is ismert PING elvhez hasonlítható. Ha a modul TRIGGER lábára magas feszültség szintet kapcsolunk 10us időtartamig, akkor az 8 darab 40 kHz frekvenciájú jelet küld ki a T-vel jelzett (transmitter) adóra. Az R-rel jelzett (receiver) vevő fogadja ezeket a jeleket, amennyiben visszaérnek. Ha nem érnek vissza, úgy a legnagyobb mérhető távolságnak megfelelő idő

múlva emeli magas feszültségszintre a modul az ECHO lábat. Programunkban a jel kiadása, és az előbb említett láb magas szintre ugrása közti időt kell mérni.

Specifikációja:

- Üzemeltetési feszültség: 5V
- Áramfelvétel: 15mA
- Működési frekvencia: 40 Hz
- Mérhető távolság: 2cm – 4m
- Mérési szög: 15°



**V. ábra, A HC-SR04 ultrahangszenzor**

#### **6.3.4 GIROSKÓP/GYORSULÁSMÉRŐ**

Ezt egy MPU6050 [18] típusú modullal valósítottam meg. Ez egy integrált, 6 tengelyes mozgáskövető rendszer. Tartalmaz 3 irányú giroszkópot, 3 irányú gyorsulásmérőt, illetve egy DMP-t (Digital Motion Processor), ami lehetővé teszi a 9 tengelyes követést. Az adatok szolgáltatása I2C protokollon keresztül működik. Az adatok pontos mérése előtt konfigurálni kell az eszközt. Ezt ugyan azon az I2C buszon tudjuk megvalósítani, mint az adatok lekérését. Míg mintavételkor regisztereket kérünk a modultól, úgy konfiguráláskor a szabadon írható regiszterek tartalmát módosítjuk.

Néhány konfigurálható regiszterek funkciója:

- Mintavételi frekvencia módosítása
- Órajel forrása I2C buszon lévő címe
- Küszöbértékek beállítása

- Szabadesés, vagy
- Mozgatás esetén

Az MPU6050 részletesebb specifikációja:

- 16 bites AD konverter
- 16 bites regiszterméret
- 16 bites kimenő adatméret
- Tápfeszültség: 3V – 5V
- Gyorsulásmérő tartománya: 2g/4g/8g/16g (az indítási konfigurálásnál beállítható)
- Giroszkóp tartománya: 250/500/1000/2000 fok másodpercenként (indításkor ez is beállítható)



**VI. ábra, Az MPU6050 modul**

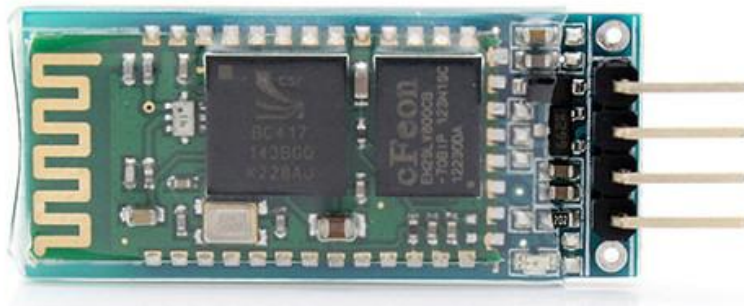
### **6.3.5 BLUETOOTH**

Ez a modul végzi a kommunikációt a két vezérlőegység között. Típusa: HC-06 [19].

Specifikációja:

- Üzemeltetési frekvencia: 3.6V – 6V (az adatlapja szerint, viszont én 3.3V-s feszültségszinten is tökéletesen dolgozom vele)Az UART protokollnak megfelelően kell az RXD (fogadó), illetve a TXD (küldő) lábára adatokat írni/fogadni.
- Az RXD és a TXD lábak 3.3V-s logikai szintet használnak

Vezérelni, konfigurálni saját beépített utasításkészletével lehet, ezek az AT parancsok. A modul TXD lábára kell a megfelelő parancsot küldeni, amíg még más eszközzel nem kapcsolódott. Ilyen parancsok lehetnek a baud rate megváltoztatása, a párosításhoz szükséges jelszó megváltoztatása, a modul neve, stb.



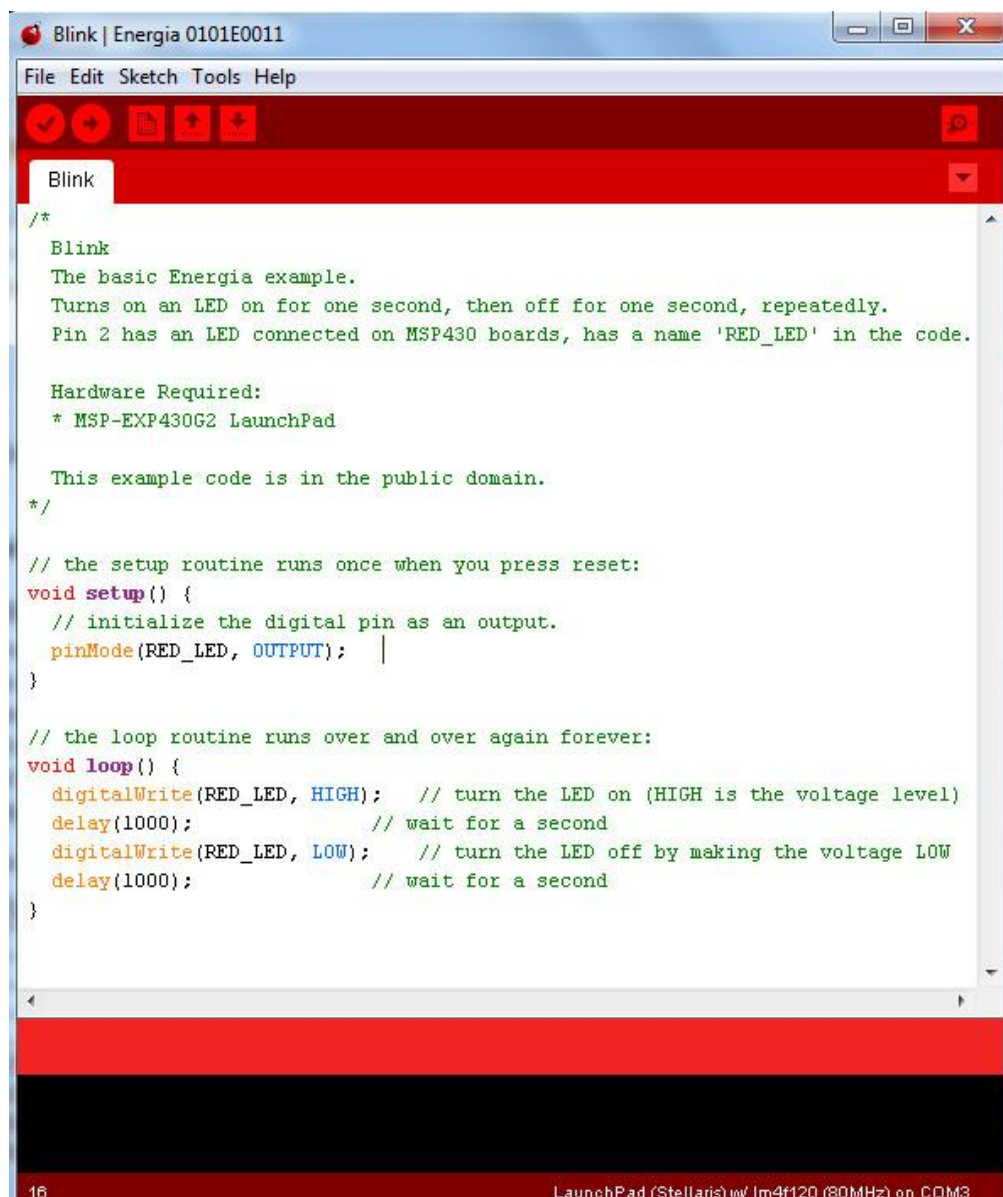
**VII. ábra, A HC-06 modul**

## **6.4 A FEJLESZTŐI KÖRNYEZET ÉS A FELHASZNÁLT**

### **PROGRAMCSOMAG**

Fejlesztői környezetnek az Energia IDE-t [20] használok. Ez egy nyílt forráskódú, elektronikai prototípus platform. Eredetijét a Texas Instruments MSP430-as kártyájához fejlesztették. A Wiring és az Arduino platformon alapú, és a Processing IDE-t használja. Az Energia nem használ bootloadert az Arduinoval szemben. A kártyán lévő ICDI, vagy FET programozható modulal kommunikál a fejlesztői eszközzel. A gyártó garantálja, hogy az Energiát ötvözve egy LaunchPad kártyával fejleszthetünk interaktív objektumokat, olvashatunk a bemeneteken a különböző szenzoroktól a kapcsolókig, sőt irányíthatunk számos fényforrást, motort, és egyéb fizikai kimeneteket. Programozni a Processing és a C++ nyelv ötvözetével lehet, amivel egy meglehetősen átlátható, könnyen szerkeszthető kódot írhatunk.

A fejlesztői környezet indítása után egy egyszerű kezelőfelülettel rendelkező ablak tárul elénk.



**VIII. ábra, Az Energia IDE**

Itt látható a fejlesztői környezet egy példaprogrammal. Az ezen a nyelven írt programoknak két fontos függvénye van, a `setup()`, illetve a `loop()`. A `setup()` függvény egy alkalommal fut le, ezért ide érdemes implementálni a modulok, illetve a GPIO lábak konfigurálását. Minden komolyabb mikrokontrollerre írt programban van egy mágikus elem, aminek még a frissen kezdő programozók nem látják értelmét. Ez a `loop()` függvény, azaz A végtelen ciklus. A fenti program a végtelen ciklust arra használja fel, hogy a `RED_LED`-re (piros LED) logikai magas jelet küld, majd vár 1 másodpercet, ami után ugyan erre alacsony jelszintet küld, majd újra

vár. Ha ezt a programot rátölti a programozó a fejlesztői kártyájára, akkor egy egyszerű LED villogó programot fog látni működés közben, 1 másodperces kitöltési idővel.

Az Energia IDE beépített függvénykönyvtára mellett felhasználtam még a Texas Instruments által kiadott kiegészítő programkönyvtárát, a StellarisWare-t. A csomag célja az, hogy gyorsítsa, és leegyszerűsítse, a Stellaris-alapú mikrokontrollerek szoftverfejlesztését.

## 7. AZ ANDROID, MINT FŐFEDÉLZETI RENDSZER

### 7.1 AZ ANDROID FEJLŐDÉSE

Az Android egy Linux alapú operációs rendszer. 2003-ban egy Android Inc. nevű cég kezdte fejleszteni, amit 2005-ben a Google vásárolt fel, és azóta is fejleszti. Az első verzió publikálására 2009-ben került sor. Ez az Android 1.5 volt és a Cupcake nevet kapta. Ez még a mai legfrissebb verzióhoz képest nagyon elmaradott volt, függetlenül attól, hogy még csak 5 év telt el a megjelenése óta. Az első szoftverfejlesztői eszköz (SDK) [21] 2007 novemberében jelent meg. Ezt a napot tartják az Android születésnapjának.

Az Android verziótörténete:

| Név                | Verziószám    | API szint | Kiadás      |
|--------------------|---------------|-----------|-------------|
| Cupcake            | 1.5           | 3         | 2009.04.30. |
| Donut              | 1.6           | 4         | 2009.09.15. |
| Eclair             | 2.0 – 2.1     | 7         | 2009.10.26. |
| Froyo              | 2.2           | 8         | 2010.05.20. |
| Gingerbread        | 2.3 – 2.3.2   | 9         | 2010.12.06. |
| Gingerbread        | 2.3.3 – 2.3.7 | 10        | 2011.02.09. |
| Honeycomb          | 3.1           | 12        | 2011.05.10. |
| Honeycomb          | 3.2           | 13        | 2011.07.15. |
| Ice Cream Sandwich | 4.0.3 – 4.0.4 | 15        | 2011.12.16. |
| Jelly Bean         | 4.1.x         | 16        | 2012.07.09. |
| Jelly Bean         | 4.2.x         | 17        | 2012.11.13. |
| Jelly Bean         | 4.3.x         | 18        | 2013.07.24. |
| KitKat             | 4.4.x         | 19        | 2013.10.31. |

Az Android eszközökre írt alkalmazások Java nyelven íródnak és a Google által kiadott Android SDK-t (Software Development Kit) használják. Az SDK biztosít a fejlesztőknek egy magasabb szintű támogatást. Az alacsonyabb szintű funkciók eléréséhez NDK-t (Native Development Kit) kell használni. Az SDK tartalmazza az API könyvtárakat (az előző

verzióval való kompatibilitás támogatására), továbbá fejlesztői eszközöket, melyek szükségesek a programok fordításához, teszteléséhez, és hibajavításhoz. A futtatást a Java SE-hez hasonlóan egy virtuális gép futtatja az operációsrendszeren. Azonban ez nem az asztali PC-n megszokott Java Virtual Machine (JVM), hanem egy Dalvik Virtual Machine nevű objektum.

A Dalvik Virtual Machine a Google által létrehozott regiszter-bázisú architektúrán alapuló virtuális számítógép, ami az Android operációsrendszereken futtatja a programkódokat. Minden elindított alkalmazás egy külön folyamatként indul el egy dedikált DVM példánnyal. Az ilyen rendszerre írt Java programkódok először Java byte kódra fordítódnak, majd ezek a VM számára is feldolgozható .dex kiterjesztésű (Dalvik EXecutable) fájlkká konvertálódnak a dx eszköz segítségével, és ezek települnek fel az eszközre. Egy .dex fájl felépítését tekintve összetettebb egy .class fájlénál. Több osztály is szerepelhet benne, a többször előforduló konstansok, melyek többször is előfordulnak különböző osztályok között, csak egyszer tárolódnak. A Dalvik Executable fájlok módosítására is van lehetőségünk az eszközre való telepítés után is. Lehetőségünk van könyvtárak, adatszerkezetek inline linkelésére, vagy a byte sorrend felcserélésére. A DVM érdekessége még, hogy rendelkezik egy futásidejű fordítóval is. Ez azt jelenti, hogy a .class fájlok nem egyből fordítódnak le, hanem folyamatosan, a program futása közben, több részletben.

Ennek utódja, a már az Android KitKatbe implementált, Android RunTime (ART). Működési elve annyiban tér el a DVM-től, hogy nem a futási időben fordítja az alkalmazásokat, hanem a telepítés ideje alatt, és csak egyszer. Ezzel energiát spórolunk, mivel a VM nem végez folyamatos fordítást.

Dolgozatomban egy 2.3.3-as Gingerbread verziót használok. Ennek alapja egy 2.6.35.7-es Linux kernel. Ez a 10-es szintű API-nak felel meg. Az előző verziókhoz képest – az előzetes elvárásokkal szembe menve – egy erős optimalizálásra került sor ebben a verzióban, néhány kisebb bővítéssel. Az optimalizálások, illetve a bővítések közé tartoznak többek között a következők:

- Multitouch támogatás
- Optimalizált energiagazdálkodás
- Hosszabb üzemidő
- WXGA kijelzők támogatása

- Újabb grafikus driver
- Frissített OpenGL ES
- NFC
- VoIP
- Szélesebb körű Bluetooth támogatás (ez elősegíti az alkalmazásokat az olyan eszközökkel való kommunikálást is, ahol nincs felhasználói felület (User Interface))

Egy Android alkalmazás mindig egy Activity objektumból kiterjesztett osztály példányosításával indul. Az Activity az absztrakció mivoltából adódóan tartalmaz előre definiált függvényeket, amiket a gyermekosztályban mindenféleképpen felül kell definiálnunk, akár üres testtel is. A fordítónak egy „@Override” annotációval kell jelezni, hogy a függvény egy kiterjesztett osztálybeli függvény felüldefiniálása. Ezt szintaktikailag a függvény hozzáférhetőségének definiálása előtt kell használnunk.

## **7.2 AZ ANDROID PROJEKT BEN HASZNÁLT KÜLSŐ**

### **PROGRAMCSOMAG**

Az Android projektben egyetlen külső programcsomagot használunk, ami a WiFi-n keresztüli hálózati kiszolgáló indításához és működéséhez szükséges. Ez a NanoHttpd [22], ami egy Modified BSD licence alatt kiadott programkönyvtár. Egy Java fájlt biztosít számunkra, aminek példányosításával egy olyan hálózati kiszolgálót hozhatunk létre, mely képes a HTML protokoll 1.1-es verzióját felhasználva kapcsolatot biztosítani a csatlakozott klienssel. Legfrissebb verziója a 2.0.5, amit 2013.12.12.-n publikáltak.

Lényegesebb core-features:

- Egyetlen Java fájlból áll
- SSL támogatás
- Nincsenek előre megfogalmazott konfigurációs fájlok, logolás
- Alapvető támogatás a cookie-k használására
- Kezdetleges PUT támogatás
- A HEAD és DELETE kérések értelmezése
- GET és POST metódusok paramétereinek elemzése

- Fájlok feltöltése (minimális maximális méret meghatározással)
- Nem tárol semmit
- Nincs sávszélesség korlát
- Összetettebb kérések esetén sem terheli le a memóriát

### 7.2.1 A HTML

A HTML (HyperText Markup Language) [23] webes dokumentumok leírására használt nyelv, az 1990-es évek eleje óta. Az ilyen dokumentumok az egyszerű szövegek mellett olyan elemeket (HTML-tag) tartalmaznak, amelyek utasítások a webböngészőknek/megjelenítő eszközöknek. Ezek körbeveszik a dokumentum egészét, vagy egyes részeit, így megadják a kliensprogramnak, hogyan értelmezze a dokumentum adott részeit. Kétfajta tulajdonságot tartalmazhatnak:

- Attribútumokat
- Konkrét tartalmat

Egy HTML-elem szintaktikailag a következő módon épül fel:

- Kezdőcímke
- Attribútumok
- Tartalom
- Zárócímke

```
<a href="http://www.google.com">Google!</a>
```

#### IX. ábra, Példa egy egyszerű linkelésre

A fenti példában egy egyszerű linkelés látható. A böngészőben egyedül a „Google!” felirat jelenik meg, amire ha rákattintunk, a href attribútumban megadott helyre ugrik a böngésző.

Dolgozatomban a HTML 4.01 verzióját használom, amit 1999-ben publikált a W3C (World Wide Web Consortium), mint ajánlás.

### 7.3 AZ ÉRZÉKELŐK MŰKÖDTETÉSE

A legtöbb Android működtette eszközben találhatóak beépített érzékelők. Ezek lehetnek elmozdulást, hőmérséklet, vagy egyéb természeti kölcsönhatást figyelő berendezések. Segítségükkel egy alkalmazás fejlesztésekor növelhetjük a végfelhasználó programhasználat közbeni élményét. Például sokkal hatásosabb egy olyan autóversenyzős játék, amiben a telefonunk dőlésével egyenes arányban fordul az általunk vezetett autó is. Nem csak játékfejlesztésre használhatóak a beépített érzékelők. Kihasználva a nagy pontosságukat, segítségükkel mobilkészülékünket át tudjuk alakítani például önálló mérőállomássá, ami képes az időjárási adatokat gyűjteni, feldolgozni, továbbítani.

Az Android platform három fő kategóriáját különbözteti meg a szenzoroknak:

- Mozgás érzékelők: Ezek a szenzorok mérik a gyorsulási és a forgatási erő nagyságát három tengelyen. Ilyen érzékelők a giroszkóp, gyorsulásmérő, gravitációérzékelők.
- Környezet érzékelők: Ezek az érzékelők a különböző környezetben vizsgálható értékeket mérik. Ilyenek például a páratartalom, légnyomás, hőmérséklet.
- Pozíció érzékelők: Az ebbe a csoportba tartozó érzékelők az eszköz fizikai helyzetét vizsgálja. Ilyen érzékelő a magnetométer.

Ahhoz, hogy ezeket használni tudjuk, az Android Sensor Framework-t [24] (Android Érzékelő Keretrendszer) kell használnunk. Néhány érzékelő hardware-alapú, mások software-alapú. Az előbb említettek tényleges fizikailag is létező hardware elemek, melyek beépítve lehetnek a mobilkészülékbe, vagy egy külső tartozékába, és környezeti értékeket mérnek. A software alapúak egy, vagy több hardware-es érzékelőből szerzik a mérési eredményeket. Ilyen például a lineáris gyorsulásmérő szenzor.

Ahhoz, hogy az 4. pontban említett folyamatos adattovábbítás a program futása alatt végig működjön, több érzékelő adatait folyamatosan kell olvasni, miközben a program esetenként fényképez, esetleg kiszolgálja a rá csatlakozó klienseket. Ezt osztálypéldányok külön szálon való futtatásával érhetjük el a leggyorsabb esetben. Az Android SDK-nak van az ehhez hasonló, egyszeri futásigényű feladatokra írt absztrakt osztálya. Ezek az `IntentService`, és a `Service` osztályok.

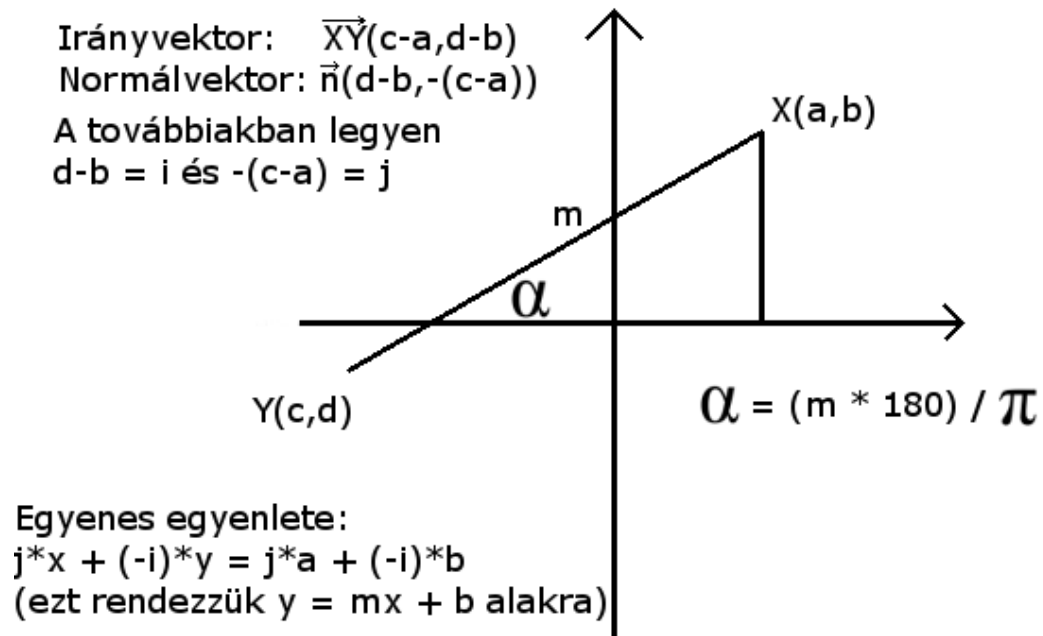
Objektumorientált programnyelveknél az absztrakt osztály tartalmaz előre megírt, és még implementálásra váró függvényeket, továbbá változókat. Az implementálandó függvények

feje definiálva van, egyedül a függvény teste üres. Például abban az esetben, amikor az osztályunk kiterjeszti az `android.app.Service` [25] osztályt, akkor implementálnunk kell (akár üres testtel is) az előírt absztrakt metódusokat. Jelen esetben ezek a következők: `onCreate()`, `onDestroy()`, `onBind(Intent)`. Az `android.app.Service` osztályban található többi függvény is szabadon felüldefiniálható. Ezt egy „@Override” annotációval tehetjük meg, amit szintaktikailag a függvény hozzáférhetőségének definiálása – amennyiben van ilyen - elé kell beírunk. A program működése közben kettő ilyen osztályt példányosít, ezek a `GPSService`, illetve a `Compass` osztályok.

A `GPSService` a GPS koordináták lekérése után Bluetooth-on keresztül egy olyan adatsomagot továbbít a Stellaris mikrokontroller felé, ami tartalmaz két lényeges információt. Egyik egy lebegőpontos szám, ami az északhoz való elfordulás nagyságát határozza meg, a másik egy string. A string lehetséges értékei: „ÉSZAK-NYUGAT”, „DÉL-NYUGAT”, „ÉSZAK-KELET”, „DÉL-KELET”. Ezt azt reprezentálja, hogy az autó a célponthoz képest merrefelé van. Az elfordulás nagyságát a következő módon számolja ki a `GPSService` osztály:

- Legyen  $X(a,b)$  az autó GPS szerinti helyzete,  $Y(c,d)$  pedig a célpont koordinátái
- Első lépésként meghatározzuk az  $XY$  egyenes egyenletét
- Ezután felírjuk a két pont irányvektorát, majd abból normálvektort számolunk
- Az egyenes egyenletét átalakítjuk  $y = mx + b$  alakra, ahol „ $m$ ” a meredekség, a  $(0,b)$  pont pedig az egyenes „ $y$ ” tengellyel vett metszéspontja
- Itt felrajzolhatunk egy derékszögű háromszöget, aminek átfogója az  $XY$  egyenes, egyik befogója az  $x$  tengely, átfogójának hossza pedig az előző pontban említett rendezett egyenletből az „ $m$ ”.
- Derékszögű háromszögre alkalmazhatjuk a szögfüggvényeket. Itt nekünk a lenti ábra szerint az  $\alpha$  szöget kell meghatározni. Ehhez a tangens szögfüggvényt kell használnunk.
- Ha ez megvan, megvan egy északhoz viszonyított szögünk. Így addig kell fordulnia a kocsinak, amíg ugyan akkora az északhoz való eltérése az iránytű szerint, mint az előbb kijött eredmény.
- A felsorolás előtt említettem, hogy elküld egy másik információt is a Stellaris mikrokontrollernek. Ahhoz, hogy az előbb említett információt megtudjuk,

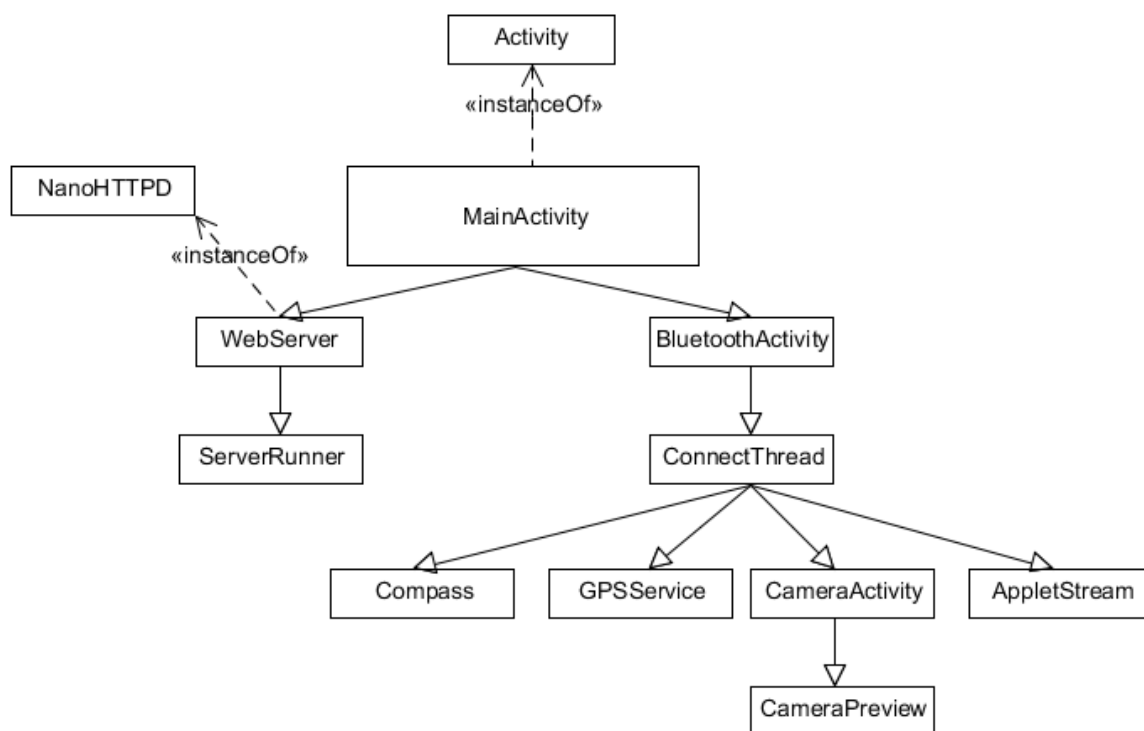
egyszerűen össze kell hasonlítani egymással az X és az Y pont koordinátáit. Például, ha a  $(a > c)$  és  $(b > d)$ , akkor ez azt jelenti, hogy az X pont koordinátái nagyobbak, mint az Y ponté, ezért az X pont az Y-hoz képest Észak-keletre van.



**X. ábra, Koordinátageometria**

A beépített iránytű aktuális helyzetét ennél könnyebb módon kaphatjuk meg. Egyszerűen egy – a már fent említett – android.app.Service-el kiterjesztett osztályt hozunk létre, majd néhány soros kalibráció után (iránytűmodul objektumának átvétele, mintavételi idő beállítása, szenzor módosulására listener állítása) már a szenzor szolgáltatja is az adatokat.

## 7.4 A PROJEKT FUNKCIONÁLIS ELEMEI RÉSZLETESEN



XI. ábra, Egyszerű UML diagram

## **8. A MEGVALÓSÍTÁS**

### **8.1 A KEZDETEK**

A megvalósítás első lépése az autó vázának, illetve a különböző moduloknak a beszerzése volt. Először a következő modulokat vásároltam meg:

- 4db ultrahangszenzor
- 1db giroszkóp/gyorsulásmérő
- Az autó váza
- Bluetooth modul
- Stellaris LaunchPad
- Próbapanel
- Kábelek

A leszállítást megelőző 3-4 hétben volt időm tanulmányozni az előző pontokban említett modulok működését, implementálását, így amikor kézhez vettem azokat, elkezdhettem a modulok működtetésének tényleges megvalósítását.

### **8.2 A TÉNYLEGES MEGVALÓSÍTÁS**

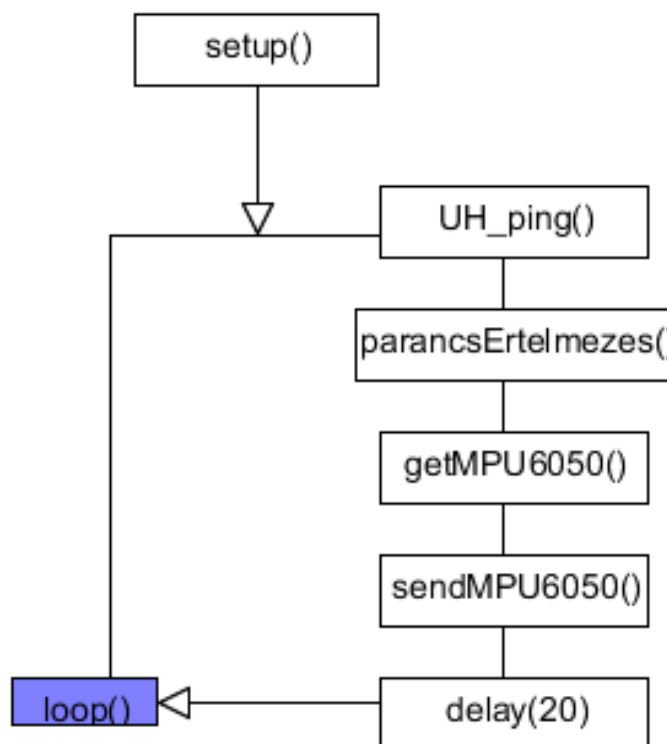
Első körben a motorvezérlést valósítottam meg. Ez egyszerű feladat volt az 5.4.2-es pontban említett H-híd IC használata. Ahhoz, hogy egy motor elinduljon valamelyik irányba, az INPUTX lábára ( $X = 1,2,3,4$ ) logikai alacsony, míg a hozzátartozó másik INPUT lábára logikai magas feszültség szintet kell kapcsolni, hogy a motor működésbe lépjen. Ugyan ezen elven a négy különálló motor vezérlését könnyen meg lehet valósítani. Előre – és hátrahaladásnál az összes motornak ugyanabba az irányba kell forognia, míg forduláskor páronként 2-2 oldalsó keréknek kell egy irányba, a másik kettőnek az ellenkező irányba.

A Stellaris LaunchPadra írt programom szerint a motorok működését két dolog határozza meg. Az ultrahangszenzorok által visszajelzett távolságadatok feldolgozása után dönt a program, hogy akadálykerülést kell-e megvalósítania, vagy egyszerűen haladnia kell-e tovább a cél irányába. Ezt a program fő végtelenciklusa első soraiban valósítottam meg. Az ultrahangszenzorokat „pingeljük”. Ezután három lehetséges dolog állhat fent.

- A cél irányába állunk, az első ultrahangszenzor nem jelez akadályt 10 cm-es távolságon belül, így haladunk tovább egyenesen.
- Nem állunk a cél irányába, de a két oldalsó ultrahangszenzor közül legalább az egyik egy tárgyat jelez a kocsi mellett legfeljebb 10 cm-es távolságon belül, abban az irányban, amerre fordulnia kellene a kocsinak, hogy célba álljon, így haladunk végig az akadály mellett addig, amíg el túl nem megyünk rajta.
- Nem áll a kocsi a cél irányába, egy ultrahangszenzor sem jelez 10 cm-en belül semmit, így a cél irányába kell fordulnia.

Ezeknek az implementálása még nem hibamentes. Ahhoz, hogy az eddig tárgyaltak működni tudjanak, szükségem volt még egy fontos változóra, ez pedig az a bizonyos irány, amerre haladnia kell az eszköznek. Ezt az iránytű modul szolgáltatja, ami viszont az Android okostelefonban van implementálva. Mivel ez fog még több adatot is szolgáltatni, ezért címkékkel láttam el az adatcsomagokat. Egy adatcsomag felépítését ezért így építettem fel: <adatcsomag jelző karakter> <érték>[, <érték>, ...]. Például azt, hogy a cél a mágneses északhoz viszonyítva hány fokra van, a következő módon továbbítja az mobiltelefon: [C 173.0], ahol C a „compass” (iránytű) szót jelöli, a 173.0 pedig az eltérést fokokban.

A kocsi önálló irányítása és a rá írt program tulajdonképpen ki is merülne ennyiben, ha nem lenne szükséges az is, hogy szolgáltatson információkat az okostelefon irányába. A szolgáltatandó adatok a giroszkóp, gyorsulásmérő és hőmérő adatok. Ezeket az MPU6050-es modulból kérhetjük le az I2C protokoll használatával. A lekérést a fő végtelenciklusban helyeztem el az előbb említett motorvezérlő algoritmus után. Itt már nem kell címkékkel ellátni a küldendő adatfolyamot, mert mindig ugyanazok az adatok továbbítódnak. Viszont valahogy jelölni kell a küldendő byte folyamat a feldolgozást megkönnyítése végett. Ezt a következő módon valósítottam meg: „255 255 255 64 0 0 42\n”, ahol az első három a giroszkóp X, Y, Z tengely szerinti elmozdulás, a második három a gyorsulásmérő X, Y, Z tengely szerinti forgatás, míg az utolsó szám a hőmérséklet értékét jelöli. A lezáró karakter egy „|”, illetve egy sortörést jelző „\n” karakter. Annak érdekében, hogy a fogadó Android eszköz ne legyen teljesen lefoglalva a folyamatos adatfogadás közben, a végtelenciklusban elhelyeztem egy olyan sort, amivel a program 20 ms-t vár az adatok szolgáltatása, illetve fogadása közben. Ennek segítségével igaz lassul az adatok szolgáltatása, viszont ez még mindig másodpercenkénti 50 adatcsomag továbbítását teszi lehetővé.



**XII. ábra, A loop() függvény felépítése**

A Stellaris mikrokontrollerre írt programom itt véget ért. Következett a mobilkészülékre írt program implementálása. Mivel ebben az időben az Android eszközök programozása elég távol állt tőlem, viszont az alapvető programozási ismereteket objektumorientált nyelvek terén már elsajátítottam, úgy gondoltam, hogy egyenesen belevágok a közepébe. Különböző példaprogramok írása helyett egyből a saját alkalmazásom megvalósításával kezdtem. Ez a módszer véleményem szerint hatásosabb volt, mivel a Java nyelv közel áll hozzám. Azokat az algoritmusokat, szenzorhasználatokat pedig, amelyekhez csak az Android SDK-ban használt objektumok használatával lehet megvalósítani (úgy, mint az Intent, IntentService, SensorManager, stb...), az Android 2.3.6-os API böngészésével sikerült implementálnom. Mivel az Android operációsrendszert futtató készülékekre írt alkalmazások mindegyikének – ahogy a Java nyelvben is ismert – szüksége van egy main (fő futtatandó) osztályra. Ehhez egyszerűen létrehoztam egy osztályt, aminek kiterjesztettem az Activity osztályt és implementáltam az absztrakt metódusokat.

A Stellaris LaunchPadról kapott adatok fogadásához szükségem volt egy Bluetooth modult menedzselő objektumra, ami a ButuActivity nevet kapta. Ehhez szintén az Activity osztály kiterjesztése és a kötelező metódusok megírása volt az első feladat, majd egy olyan külön szálon indított osztályra volt szükségem, amely elvégzi a tényleges csatlakozást a Stellaris LaunchPadhoz csatlakoztatott Bluetooth modulhoz. Ehhez létrehoztam a ConnectThread osztályt, kiterjesztve a Runnable absztrakttal. Azt, hogy mi fusson egy külön szálon, azt a Runnable osztály run() absztrakt metódusa felüldefiniálásába implementáltam. Míg az osztály konstruktorában valósul meg a csatlakozás, az előbb említett függvényben a tényleges adatfogadás. Az adatok másik irányba való áramlását (a Stellaris LaunchPad felé) megoldva definiáltam két függvényt. Ezek a send(int) és sendStr(String) metódusok. Előbbivel egy egyszerű integer típusú számot tudunk továbbítani neki, utóbbival pedig egy egész stringet. A sendStr(String) függvénnyel valósul meg az adatcsomagok továbbítása.

A Stellaris LaunchPad és az Android okostelefon között cserélt adatokat meg is kell jeleníteni valahol. Ezt első elgondolás szerint egy PC kliensprogram valósította volna meg, ehhez viszont folyamatosan magunknál kell tartani a programot, ha meg szeretnénk nézni a megjelenített adatokat. Ehelyett egy hálózati kiszolgálót implementáltam egy Java nyelven írt külső programcsomagot felhasználva, ami egy HTML alapú szervert biztosít. Ezzel elértem azt, hogy ha a mobilkészülékkel csatlakozunk egy vezeték nélküli hálózathoz, vagy az Android operációsrendszer „Mobil hozzáférési pont” funkcióját használjuk, és arra csatlakozunk fel a kliensünkkel, akkor a mobiltelefon IP címére csatlakozva egy böngészőklienssel egy olyan weboldal tárul elénk, ami folyamatosan mutatja az érzékelők és modulok aktuális állapotát. A következő modulok állapotát tekinthetjük meg itt:

- Kamera
- Giroszkóp
- Gyorsulásmérő
- Hőmérséklet
- GPS

Ahogy fent említettem, egy Java nyelven írt programcsomagot felhasználva hoztam létre ezt az osztályt. Ez a NanoHttpd, aminek segítségével csak a port számát kell megadnunk, a HTML szerver indításához. Természetesen ezt is egy külön szálon indítottam.

A kamera állapotát 2 másodpercenkénti képfrissítéssel tekinthetjük meg a szerveren. Mondható folyamatos video streamelésnek 0.5 FPS-sel (Frame Per Second). Az emberi szem által is szép, mozgóképeknek tűnő videó kép továbbítását értelmetlennek láttam, mivel az eszköz nagyon lassan mozog. A fényképezést a CameraActivity osztályban valósítottam meg, ami mellé kellett egy képmegjelenítő is támogatva azt a felhasználói beavatkozást, hogy úgy helyezhessük el a kamerát a kocsin, ahogy szeretnénk, és erről egyfajta visszajelzést is kapjunk. Ez a CameraPreview osztályban van implementálva. Maga a fénykép készítését a CameraActivity osztályban írtam meg. Egy külön szálon indít egy végtelen ciklust, amiben a fényképezés két másodpercenként van megvalósítva. Minden két fényképezés között a kép eltárolódik az Android eszköz memóriakártyájára, ahonnan a hálózati kiszolgáló ki tudja olvasni, majd továbbítani a böngészőkliens irányába. A minél gyorsabb fényképezés érdekében a telefon kamerája nem fókuszál, nem használ speciális effekteket.

A GPS koordináták eléréséhez egy a már 6.3-as pontban említett Android SDK-beli Service osztályt kellett kiterjesztenem. Ezt a GPSService osztályban valósítottam meg. Így futás közben egy olyan objektumot hoz létre a program, ami a háttérben fut és mindenféle felhasználói interakciótól mentes. Egyetlen dolga az, hogy adatokat kérjen a GPS modultól egy előre konfigurált időközönként, majd meghívva a ButuActivity egy függvényét, továbbítsa azt vele a Stellaris LaunchPad irányába. A modultól való lekéréséhez implementálnom kellett továbbá az Android SDK-beli LocationListener osztályt a GPSService-ben. Ezzel lehetőségem vált olyan függvénytestek megírására, mint például az onLocationChanged(Location), ami akkor fut le, ha változott a helyzetünk. Az osztály konstruktorában az azon kívül létrehozott LocationManager típusú objektumot konfiguráltam, aminek négy paramétert kellett átadnom, melyek a következők:

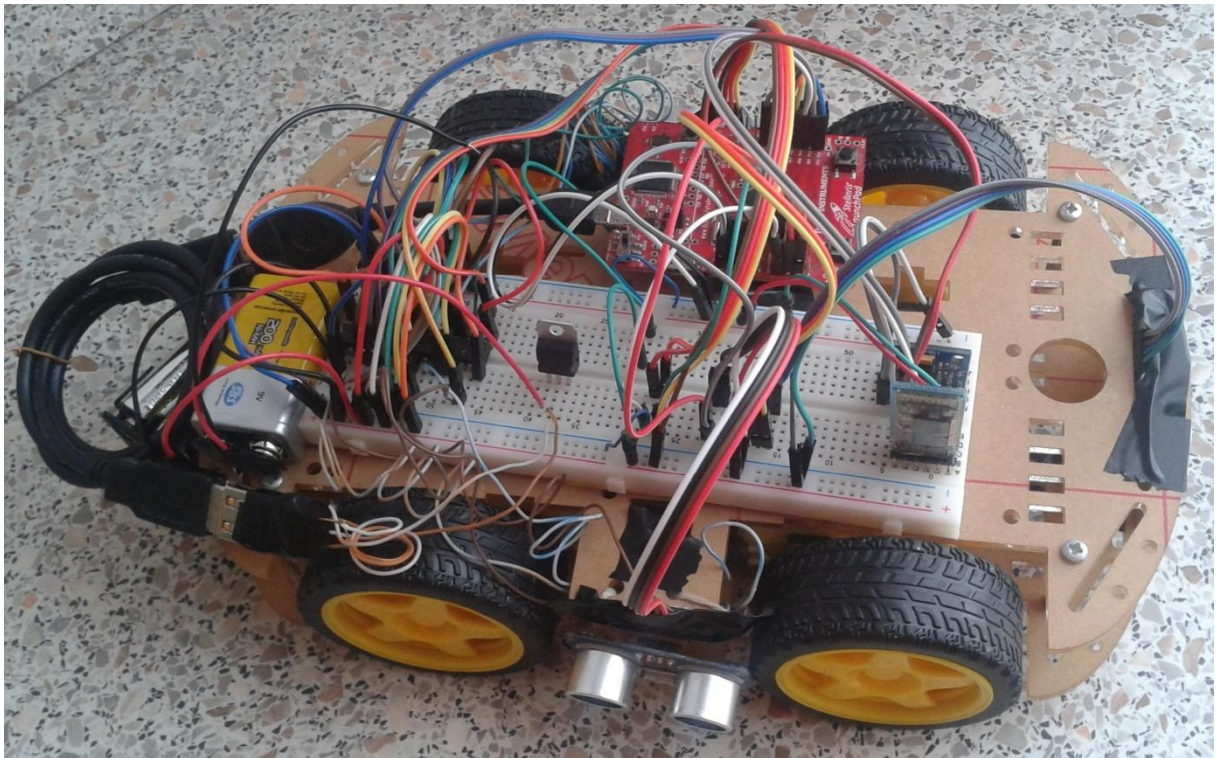
- Egy string, amivel beállítjuk, melyik modul legyen az ellátó. Itt a LocationManager osztály egy konstansát használtam fel, a LocationManager.GPS\_PROVIDER-t. Ezzel a GPS modulból veszi az adatokat.
- Második paraméter egy egész szám, amivel megadtam neki, hogy 10 másodpercenként frissítse az adatokat. Ez soknak tűnik, viszont ha a kocsin 0.8 métert tesz meg másodpercenként, így már véleményem szerint ez ideális.

- Harmadik paraméter egy lebegőpontos szám, ami azt adja meg, hogy mekkora az a minimum távolság, amire frissülnek az adatok. Ezt az argumentumot 1 méternek választottam meg.
- Végül az utolsó paraméter egy `LocationListener` típusú objektum, amely `onLocationChanged(Location)` metódusa meghívódik minden frissítéskor. Mivel úgy hoztam létre a `GPSService` osztályt, hogy kiterjesztettem vele a `LocationListener`-t, így használhattam azt utolsó paraméternek.

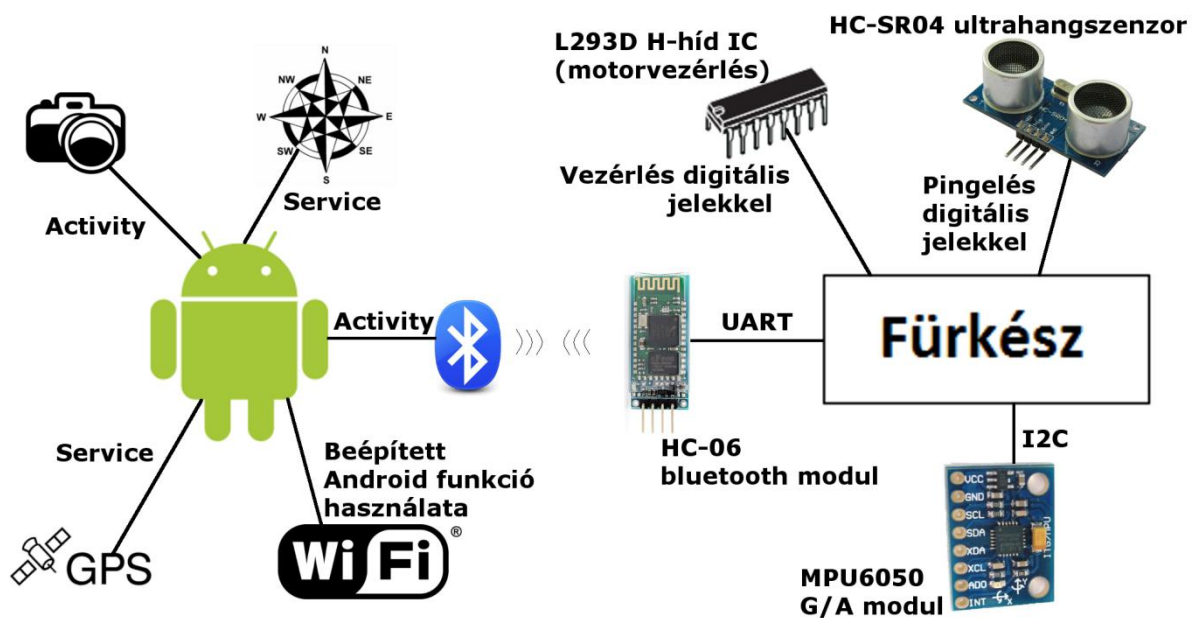
A beépített iránytű adatainak lekérésére a `Compass` osztályban – ahogy az előbb említett `GPSService`-ben - kiterjesztettem a `Service`-t. Ebbe egy olyan listenert regisztráltam, ami két másodpercenként frissíti az iránytű állapotát. Ez megint sok időnek tűnhet, mivel az eszköz folyamatos mozgásban van. Viszont az eszköz csak álló helyzetben fordul, és körülbelül 3 másodperc alatt fordul 90 fokot, így ez ideális választás véleményem szerint.

A giroszkóp, illetve gyorsulásmérő adatait terv szerint különböző Java nyelven írt Appletek [33] szolgáltatnák a felhasználónak a webes felületen. Eddig még csak egyet sikerült implementálnom, ez pedig a gyorsulásmérő X tengely szerinti elmozdulását figyeli. Az Applet egy egyszerű koordináta-rendszerben folyamatosan jeleníti meg az aktuális értékeket mindig eltolva a legfrissebb értékkel. Így az elmúlt 20-25 másodperc állapotát vissza tudjuk nézni. Mivel az ez a böngészőkliensen fut, ezért nem kap direkt információkat. Ezt úgy oldottam meg, hogy az okostelefon Socketeket használva kapcsolódik a böngészőn futó Applet-hez, folyamatos adatáramlással szolgáltatva neki a legfrissebb információkat.

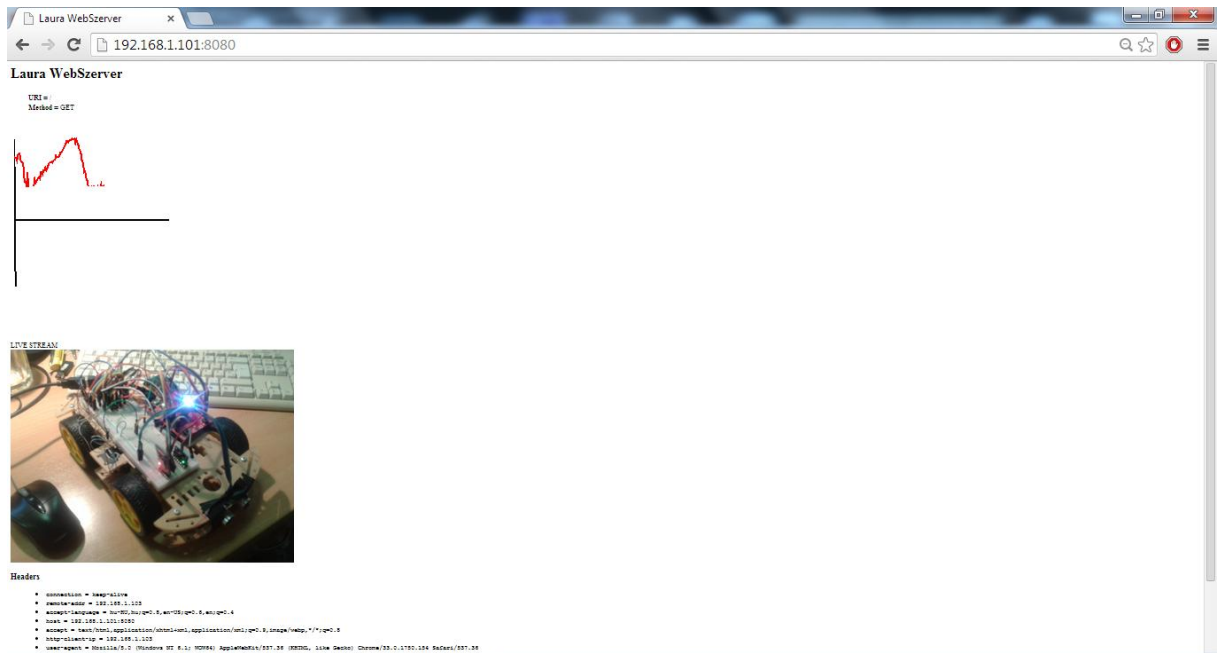
Néhány kép a kész eszközről:



XIII. ábra, a kész eszköz oldalnézetben



XIV. ábra, Az implementált hardveres és szoftveres modulok kommunikációja



**XV. ábra, Az okostelefon által létrehozott mobil hozzáférési pont böngészőkliensből nézve**

## 9. ÖSSZEFOGLALÁS

Dolgozatom keretein belül készítettem egy önálló mozgásra képes robotautót. Az autó alapját képező vázat előre megvásároltam. Ez egy négy motorral (kerekenként 1-1) ellátott tank-szerű mozgásra alkalmas kocsi. Ennek az irányító fedélzeti rendszerét egy Stellaris LaunchPad alkotja. Ehhez kapcsoltam még különböző modulokat, mint például ultrahangszenzor, Bluetooth modul, illetve egy okostelefon amelyek a kommunikációt és a tájékozódást segítik. Ezen felül megalkottam egy kommunikációs protokollt is hozzá, amely segítségével az előbbi modulok adatokat tudnak szolgáltatni a fejlesztői kártya, illetve az e fölé helyezett Android eszköz között. Utóbbiban implementáltam egy hálózati kiszolgálót is, aminek segítségével az arra csatlakozott kliensek a különböző szenzorok adatai nyomon követhető.

Az eszköz jelenlegi állapota szerint a Stellaris LaunchPadra csatlakoztatva vannak az ultrahangszenzorok, a Bluetooth modul és a giroszkóp/gyorsulásmérő. A Bluetooth modul kommunikál az Android eszközzel, így adatcserére képesek. Az okostelefonon futó hálózati kiszolgálón pedig képes megjeleníteni a mobilkészülék beépített moduljainak és az irányító fedélzeti rendszerről kapott adatokat.

A közeljövőben terveim közé tartozik a további érzékelők beszerelése, továbbá a rendszer függetlenítése az autó vázától, és ezzel áthelyezve azt egy repülésre képes eszközre.

## **10. IRODALOMJEGYZÉK**

(Az itt található összes hivatkozás utolsó elérési ideje 2014.05.04.)

[1] A Vasember film

<http://www.imdb.com/title/tt0371746/>

[2] RUR

<http://jerz.setonhill.edu/resources/RUR/>

[3] DVI 2860 szabvány

[http://www.vdi.eu/guidelines/vdi\\_2860-montage\\_und\\_handhabungstechnik\\_handhabungsfunktionen\\_handhabungseinrichtungen\\_begriffe\\_definitionen/](http://www.vdi.eu/guidelines/vdi_2860-montage_und_handhabungstechnik_handhabungsfunktionen_handhabungseinrichtungen_begriffe_definitionen/)

[4] Robotika három törvénye

[http://hu.wikipedia.org/wiki/A\\_robotika\\_h%C3%A1rom\\_t%C3%B6rv%C3%A9nye](http://hu.wikipedia.org/wiki/A_robotika_h%C3%A1rom_t%C3%B6rv%C3%A9nye)

[5] Unimate

<http://www.robotalloffame.org/inductees/03inductees/unimate.html>

[6] John Hopkins Beast

<http://www.frc.ri.cmu.edu/~hpm/talks/revo.slides/1960.html>

[7] Curiosity marsjáró robot

<http://mars.jpl.nasa.gov/msl/>

[8] A NASA 2012-es bejelentése

[http://index.hu/tudomany/2012/12/05/a\\_mars-kutatas\\_legizgalmasabb\\_evei\\_jonnek/](http://index.hu/tudomany/2012/12/05/a_mars-kutatas_legizgalmasabb_evei_jonnek/)

[9] Google self-driving car

<http://spectrum.ieee.org/automaton/robotics/artificial-intelligence/how-google-self-driving-car-works>

[10] Stellaris LaunchPad

<http://www.mouser.com/ds/2/405/lm4f120h5qr-124014.pdf>

[11] Az Android operációsrendszer

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

[12] Mikrokontroller

<http://en.wikipedia.org/wiki/Microcontroller>

[13] UART átvitel szemléltetése

<https://www.freebsd.org/doc/en/articles/serial-uart/>

[14] I2C átvitel szemléltetése

[http://esca.atomki.hu/PIC24/spi.html#Az\\_I2C\\_kommunik%C3%A1ci%C3%B3s\\_csatorna](http://esca.atomki.hu/PIC24/spi.html#Az_I2C_kommunik%C3%A1ci%C3%B3s_csatorna)

[15] DC motorok

[http://hu.wikipedia.org/wiki/Kefe\\_n%C3%A9lk%C3%Bcl\\_egy%C3%A9n%C3%A1ram%C3%BA\\_motor](http://hu.wikipedia.org/wiki/Kefe_n%C3%A9lk%C3%Bcl_egy%C3%A9n%C3%A1ram%C3%BA_motor)

[16] H-híd kapcsolási rajz

<http://www.talkingelectronics.com/projects/H-Bridge/H-Bridge-1.html>

[17] HC-SR04 modul

<http://www.micropik.com/PDF/HCSR04.pdf>

[18] MPU6050 modul

<http://www.invensense.com/mems/gyro/mpu6050.html>

[19] HC-06 modul

[http://makezineblog.files.wordpress.com/2014/03/hc\\_hc-05-user-instructions-bluetooth.pdf](http://makezineblog.files.wordpress.com/2014/03/hc_hc-05-user-instructions-bluetooth.pdf)

[20] Energia

<http://www.energia.nu/>

[21] Android SDK

<http://developer.android.com/sdk/index.html>

[22] NanoHttpd

<https://github.com/NanoHttpd/nanohttpd>

[23] HTML

<http://www.w3c.hu/forditasok/sikos/bevezet/start.htm>

[24] Android Sensor Framework

[http://developer.android.com/guide/topics/sensors/sensors\\_overview.html](http://developer.android.com/guide/topics/sensors/sensors_overview.html)

[25] Android Service osztály

<http://developer.android.com/reference/android/app/Service.html>

[26] Java Applet

<http://docs.oracle.com/javase/tutorial/deployment/applet/>

## **Felhasznált képek:**

[I] Stellaris LaunchPad és az LM4F120H5QR MCU kép

<http://www.mathworks.com/matlabcentral/fileexchange/screenshots/8494/original.jpg>

[II] Az UART működése kép

<http://m.eet.com/media/1076583/MaximSoftUartFig1.jpg>

[III] Az I2C feszültség szintjei kép

[http://esca.atomki.hu/PIC24/images/ch10/I2C\\_data\\_transfer.png](http://esca.atomki.hu/PIC24/images/ch10/I2C_data_transfer.png)

[IV] Az L293D modul helyes bekötése kép

[http://www.elektrobot.hu/items/IC\\_motorvezerlo\\_L293D\\_1.jpg](http://www.elektrobot.hu/items/IC_motorvezerlo_L293D_1.jpg)

[V] A HC-SR04 ultrahang szenzor kép

[https://docs.google.com/document/d/1Y-yZnNhMYy7rwhAgyL\\_pfa39RsB-x2qR4vP8saG73rE/edit](https://docs.google.com/document/d/1Y-yZnNhMYy7rwhAgyL_pfa39RsB-x2qR4vP8saG73rE/edit)

[VI] Az MPU6050 modul kép

<http://playground.arduino.cc/uploads/Main/mpu-6050.jpg>

[VII] A HC-06 modul kép

<http://fasttechcdn.com/products/112/1129201/1129201-1.jpg>

## **11. KÖSZÖNETNYILVÁNÍTÁS**

Szeretném megköszönni témavezetőmnek, Dr. Végh Jánosnak a megbeszélések folyamán átadott hasznos ötleteket, illetve tanácsokat és a megbeszélések folyamán a helyes útra terelt.

Köszönöm szüleimnek és testvéreimnek a mindenféle támogatást és a nélkülözést, ami nélkül ez a dolgozat nem valósulhatott volna meg.

Szeretném megköszönni barátaimnak is, akik a dolgozatom elkészítésében segítettek, és mindenben támogattak. Továbbá hálával tartozom nekik azért is, amiért nem hagytak magamra sosem, és mindig megmutatták a közösségi élet szépségeit, amikor már majdnem feladtam. E nélkül valószínűleg sokkal hamarabb is befejezhettem volna ezt a dolgozatot.