

Debreceni Egyetem
Műszaki Kar
Villamosmérnöki és Mechatronikai Tanszék



TDK dolgozat

DaNI robot vezérlése kézmozgás segítségével

Készítette: **Tóth Attila János mechatronikai mérnök hallgató, 3. évf.**

Belső konzulens: Dr. Husi Géza főiskolai docens

Tanszékvezető: Dr. Husi Géza főiskolai docens

Debrecen
2011

Tartalomjegyzék

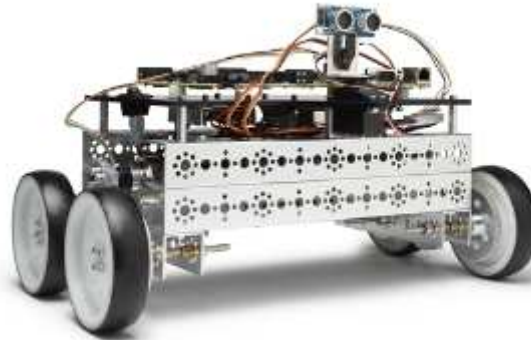
Bevezetés.....	3
1. DaNI Robot	4
1.1. A robot mozgatása	5
2. Bluetooth	6
3.1. BTM-112 Bluetooth modul.....	8
3.2. Soros kommunikáció	8
3.3. Bluetooth modul beállítása - AT parancsok	9
3.4. Bluetooth – RS232 átalakító.....	10
4. Mikrovezérlős távirányító áramkör.....	12
4.1. Adatfolyam iránya	12
4.2. Gyorsulásmérő modul.....	13
4.3. Rádiófrekvenciás modulok	14
4.4. Az áramkör megvalósítása	16
4.5. Az áramkör működése	20
4.6. Analóg-digitál konverter	20
4.7. ADC átalakítás	21
4.8. Elem feszültségének mérése.....	22
4.9. Időzítő beállítása	23
4.10. USART modul.....	24
4.10.1. USART beállítása.....	24
5. Mobiltelefonos távirányító	26
5.1. Symbian operációs rendszerre írt vezérlő.....	26
5.2. Miért Python?	28
5.3. A program működése:.....	28
5.4. Android operációs rendszerre írt vezérlő	29
5.5. Android program fejlesztése	30
6. DaNI vezérlő programja	33
7. Fejlesztési lehetőségek	37
Összefoglaló.....	38
Köszönetnyilvánítás.....	39
A témában született tudományos cikkeim	39
Irodalomjegyzék	40

Melléklet.....	42
Mikrovezérlős áramkör programja.....	42
„A” áramkör (rf868_2.c)	42
„B” áramkör (rf868.c)	46
Robotvezérlő program Symbian operációs rendszerre (DaNI control.py)	49
Robotvezérlő program Android operációs rendszerre (DaNI control.java)	50

A robotika nagyon fontos szerepet tölt be a mechatronikán belül. Ez a terület magába foglalja a jelfeldolgozást, adatgyűjtést, mechanikai hajtást, kommunikációt, számítógépes megjelenítést, vezérlést és még számos egyéb feladatot. A DaNI névre hallgató, National Instruments által gyártott robot ilyen feladatok elvégzésére alkalmas. Ebben a dolgozatban a robot mozgásának irányítására kerestünk alternatív megoldást. Az ötlet egy korábbi TDK dolgozatból ered, ahol mobiltelefon segítségével, Bluetooth kapcsolaton keresztül lehetett irányítani egy Lego Mindstorms NXT robotot. Látna az NXT robottal elért sikereket, ez az irányítási mód került átültetésre a National Instruments robotjára. Továbbfejlesztve az ötletet, ehhez a robothoz kétkézes irányító eszköz is készült. Az NXT-nél alkalmazott kommunikáció Bluetooth-on keresztül történt. A Bluetooth elterjedtsége és platformfüggetlensége miatt ebben az alkalmazásban is jó választásnak bizonyult. Mivel a robot gyárilag nem rendelkezik Bluetooth vevővel, külső áramkör csatlakoztatására volt szükség. Ezzel a módszerrel fokozatmentes sebességszabályozás, gyors irányváltás, pontos pozícionálás valósítható meg.

1. DaNI Robot

Az NI Robotics kezdő készlete, a DaNI robot az alábbi eszközökből épül fel: robotváz, kerekek, fogaskerék hajtás egyenáramú motorral, akkumulátor, SB Rio 9632-es panel, ultrahangos távolságmérő és az ezt mozgató szervo motor [1].



1. ábra: DaNI robot

Az SB Rio 9632-es panel tartalmaz egy valós idejű processzort, egy FPGA áramkört valamint egy analóg és digitális IO egységet. A robot Ethernet felületen keresztül kommunikál a számítógéppel. Ezen keresztül történik a robot programozása, valamint ezzel tartja a kapcsolatot a számítógépen futó program a robot programjával. Ezáltal futás közben figyelhetjük a robot programját, valamint be is avatkozhatunk, ha szükséges. Az adatok számítógépes megjelenítésével (akár futás közben is) a szoftverfejlesztés és hibakeresés hatékonyabbá válik.

A robotban futó programot a National Instruments szoftverével, a LabView segítségével készíthetjük el. A LabView egy grafikus programozási nyelv. Eltérő gondolkodásmódot kíván meg a procedurális és az objektum orientált programozási nyelvekhez képest. A program futását itt az adatfolyam határozza meg. Ez annak köszönhető, hogy ezt a nyelvet főként virtuális műszerezésre használják [2]. Ennek megfelelően az érzékelőből vagy más forrásból származó adat beolvasása után feldolgozásra kerül, majd ezután valamilyen kijelzés illetve beavatkozás történik. A program futását tehát a bemenő adat határozza meg. Példaként, egy függvény akkor fut le, ha minden bemenő paramétert megkap. Ennek köszönhetően a műveletek párhuzamosan is történhetnek, amennyiben nem függenek egymástól. A robot programozásához kiegészítő szoftvereket is kell telepíteni. Ilyen a LabView FPGA modul, mely az FPGA áramkör programozásához szükséges, a LabView Real Time modul, amely a valós idejű processzorhoz kell, valamint a LabView Robotics kiegészítő. Ez utóbbi szoftveren keresztül csatlakozhatunk a robothoz és tölthetjük rá a megírt programot. Tartalmaz továbbá a robotika területéhez tartozó, előre megírt függvényeket (PID hajtás, akadály elkerülés,

útvonaltervezés, stb.) Hasznos kiindulópont a mintaprogram, amely elérhető a modul indításakor. Ez a program tartalmaz egy előre elkészített FPGA bitfile-t, amely megvalósítja az FPGA áramkör segítségével a motorok sebességének szabályozását, a szervo motor pozicionálását és az ultrahangos távolságmérést. Ennek köszönhetően a programozónak nem kell foglalkoznia az alacsony szintű programozással, időzítésekkel, mert ezek készen rendelkezésre állnak. Ezzel a szoftverfejlesztés gyorsabbá válik.

1.1. A robot mozgatása

A robot mozgásának sebességét és irányát a kéz térbeli állása határozza meg. Ennek mérésére egy mikrovezérlő alapú áramkört, majd mobiltelefont használunk. A mai mobil készülékekben szinte kivétel nélkül található gyorsulásmérő áramkör. Ezt kihasználva semmilyen külső kiegészítő eszközre nincs szükség a robot mozgásának meghatározásához. Két különböző operációs rendszerű készülékre készült el a robot vezérlő programja: Symbianra és Androidra. A mikrovezérlő alapú áramkör két részből épül fel. Ebben az esetben kétkezes irányítás valósul meg. A két áramkör együttes helyzete határozza meg a mozgást. Az alkalmazott mikrovezérlő az Atmel által gyártott, AVR családhoz tartozó ATmega8, 8 bites mikrovezérlő. A gyorsulásmérő pedig a Freescale Semiconductors által gyártott MMA7361 típusú IC. A robot és az eszközök közti kommunikáció Bluetooth kapcsolaton keresztül történik. Mivel a robot nem rendelkezik Bluetooth egységgel, viszont soros portja van, egy Bluetooth-RS232 soros átalakító építésére volt szükség. Ezen keresztül a robot képes bármilyen készülékkel kapcsolatot létesíteni, amelyik rendelkezik Bluetooth adapterrel.

A három eszköz közül legkisebb számítási és memória kapacitással a mikrovezérlős távirányító rendelkezik. Ezért az összes távirányító feladata annyi lesz, hogy a gyorsulásmérőből kiolvasott nyers adatokat továbbítsa Bluetooth-on a robot felé. A többi munkát, átalakítást és számításokat a robot végzi. Ezen felül a kommunikációba kódolt egyéb adatokból a robot tudja, hogy éppen milyen eszközzel kommunikál és hogy milyen formátumban kapja az adatokat. A robot programja ennek függvényében az aktuális távirányítóhoz tartozó programrészt hajtja végre.

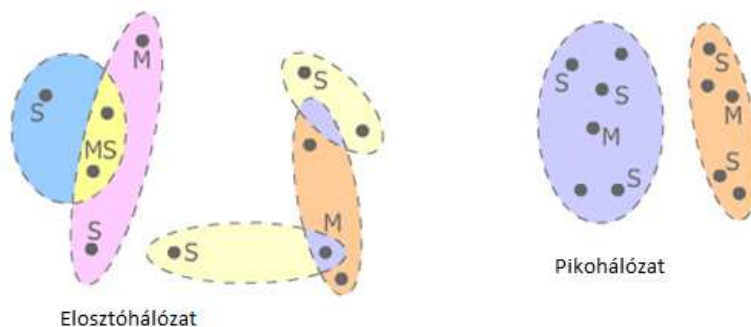
2. Bluetooth

A Bluetooth adatcserére használt vezeték nélküli szabvány. Frekvenciasávja a világszerte szabadon elérhető 2,4 GHz, amelyen belül, néhány ország kivételével (pl. Japán és Franciaország) 79 db 1 MHz-es csatornákon dolgozik (2,402-2,480 GHz). A kommunikáció ezeken a csatornákon véletlenszerűen ugrálva történik, ezáltal biztosított több eszköz egyidejű kommunikációja valamint a zavarcsökkentés is. Az adatátviteli sebessége az 1.2-es verziónál 1 Mbps, a 2.0-ásnál 3 Mbps, a 3.0-ás verziónál pedig már 24 Mbps [3]. Elkészült a Bluetooth 4.0-ás specifikációja is, ez azonban még nem jelent meg. Amit érdemes róla tudni, hogy visszafelé kompatibilis a korábbi specifikációkkal. Két üzemmódban használható: Bluetooth High Speed módban képes a 3.0-ás specifikációnál elért 24 Mbps-os sebességre, valamint Single Mode Low Energy üzemmódban akár egy évig is képes folyamatosan kommunikálni egy gombkelemtől táplálva. Természetesen ennek ára, hogy a sebesség legfeljebb 1 Mbps, ami a gyakorlatban még kevesebb lehet [4].

A Bluetooth eszközöket három csoportra oszthatjuk hatótáv szempontjából:

1. osztály: 100 m
2. osztály: 10 m
3. osztály: 1 m

Minden eszköznek egyedi, 48 bites címe van. A Bluetooth kapcsolat mester-szolga alapú, azaz egy mesterhez egyidejűleg 7 szolga eszköz csatlakozhat. A szolga csak a mesterrel kommunikálhat és csak akkor, amikor a mester engedélyezi. Az így kialakuló hálózatot pikohálózatnak nevezzük. A Bluetooth rendszer 10 pikohálózat egyidejű működését teszi lehetővé. Egy egység több pikohálózathoz is tartozhat, több hálózatban lehet szolga, de csak egy hálózatban lehet mester.



2. ábra: Bluetooth hálózatok [3]

A korábban leírtak csupán az adatátvitel fizikai megvalósítását határozzák meg. A Bluetooth specifikációk ezen felül úgynevezett Bluetooth protokollokat is tartalmaznak, amelyek megkönnyítik a Bluetooth használatát különböző alkalmazási területeken. A 3. ábrán látható a Bluetooth néhány protokolljának egymásra épülése:



3. ábra: Bluetooth protokollok [5]

Az alsóbb rétegekben találhatók azok a protokollok, amelyek minden alkalmazásban működnek (ezek a Bluetooth fizikai rétegei) pl: Baseband, ACL, SCO. Ezen belül a Link Manager protokoll gondoskodik az eszközök kapcsolódásáról, a titkosításról és a hitelesítésről. Az L2CAP (Logical Link Control and Adaptation Protocol) tartja a kapcsolatot az alsó és középső rétegek között. Feladata a Bluetooth sajátosságainak elrejtése, ezáltal támogatva a platformfüggetlenséget. A középső rétegben találhatóak a speciális felhasználói alkalmazásokat támogató protokollok. Az SDP (Service Discovery Protocol) a környező Bluetooth eszközök szolgáltatásainak felderítésére szolgál, azaz kideríti, hogy az ábrán szereplő protokollok közül melyek érhetőek el az eszközön. Az OBEX bináris adatok átadására szolgál, amelyet például fájlátvitelre használhatunk. Erre más protokollok is épülnek. Az egyik legfontosabb az SPP (Serial Port Profile) amely virtuális soros port emulációra szolgál és az RFCOMM-ra (Radio Frequency Communication) épül. Erre is számos magasabb szintű protokoll épül. A robottal való kommunikáció során ezt használjuk.

3.1. BTM-112 Bluetooth modul



4. ábra: BTM-112 modul [6]

A 4. ábrán látható egy 2.0-ás specifikációjú, másodosztályú (10 m hatótáv), SPP protokollt támogató Bluetooth modul. Ez az eszköz csak egy Bluetooth adapterrel tud kommunikálni az SPP protokoll miatt. Mind mester, mind szolga módban működik. A modul az áramkörhöz soros porton kapcsolódik. Ezen keresztül történik az eszköz vezérlése és az adatkommunikáció is. Bár ki van alakítva rajta az USB és az SPI port is, ezek csak firmware frissítésre és lekérdezésre szolgálnak. Táplálása 3,3 V-ról történik, nem TTL toleráns, azaz nem működtethető 5 V-ról. Az eszközt AT parancsokkal vezérelhetjük.

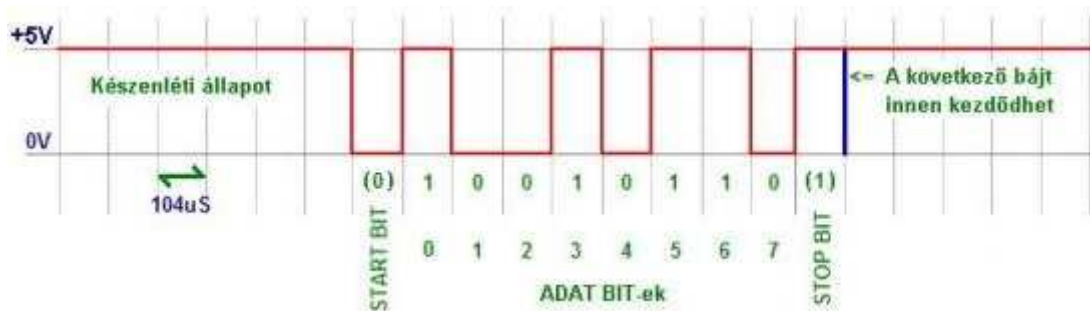
3.2. Soros kommunikáció

Soros kommunikáció alkalmával az átvinni szándékozott adatok bitjeit egymás után, sorosan juttatjuk el egyik eszköztől a másikhoz. Két fajtája ismert és használt: soros szinkron (SPI) és soros aszinkron (UART). [7]

Az SPI kommunikáció mester-szolga alapú, a kommunikáció a mester eszköz által generált órajellel szinkronizált. A buszon több szolga eszköz is lehet, melyek közül a mester egy külön vezetéken engedélyezi az eszközt, amellyel éppen kommunikálni kíván.

Az UART kommunikációban két eszköz vesz részt, melyek egyenrangúak. Itt nincs szinkronizáló órajel, vagyis az adatátvitel értelmezése az időzítésen alapul. Az információcsere két vezetéken zajlik: minden eszköznek van RX és TX lába. Az RX láb az eszköz bemenete, a TX pedig a kimenete. Az első eszköz TX lábát a második eszköz RX lábára kell kötni és fordítva. A sikeres kommunikációhoz elengedhetetlen hogy

mindkét eszköz azonos átviteli sebességre legyen beállítva, máskülönben a kommunikáció nem értelmezhető.



5. ábra: Soros adatátvitel [8]

Vegyünk például a 9600 bps-os sebességet(5. ábra). A kommunikáció mikrovezérlő esetén TTL szinten történik, azaz a logikai 1-esnek 5 V, a 0-nak 0 V felel meg. Mivel a sebesség 9600 bps, egy bit elküldéséhez szükséges idő $1/9600$, azaz 104 µs. Ha az egyik eszköz adatot kíván küldeni, a készenléti állapotból (5 V) a TX lábat 0 V-ra kapcsolja 104 µs-ig, majd elküldi az adatbiteket (esetünkben 8 db-ot). Ezután egy paritás bitet küld, amely az átvitt csomag hibaszűrését szolgálja (esetünkben ez nem használt) és végül egy stop bitet, amely lezárja a csomagot. Ebből látszik, hogy a kommunikáció megkezdése előtt a két eszközt ugyanazokkal a beállításokkal kell ellátni (átviteli sebesség, bitek száma, paritás bit használata, stop bitek száma, stb.) különben a kommunikáció sikertelen lesz.

3.3. Bluetooth modul beállítása - AT parancsok

Az eszköz adatlapja tartalmazza az AT parancsokat [6]. Soros porton elküldjük az eszköznek az „AT” karaktereket, majd a kiválasztott utasítás betűjelét és paramétereit (pl: eszközök keresése: „ATF”, csatlakozás eszközhöz: „ATA” stb.). A modulokat úgy kell beállítani, hogy a soros kommunikáció paraméterei egyezzenek a mikrovezérlőjével illetve a robotban beállított paraméterekkel. Be kell állítani a PIN kódot, és a modul szerepét a kommunikációban (vevő esetén slave, adó esetén master). A Bluetooth modul első indítása a gyári alapértelmezett beállításokkal történik. A modul szolga módban van, soros port sebessége 19200 bps. Ezek a beállítások megfelelnek a robot vevő egysége esetén. A mikrovezérlő és a Bluetooth modul között a kommunikáció sebessége 9600 bps.

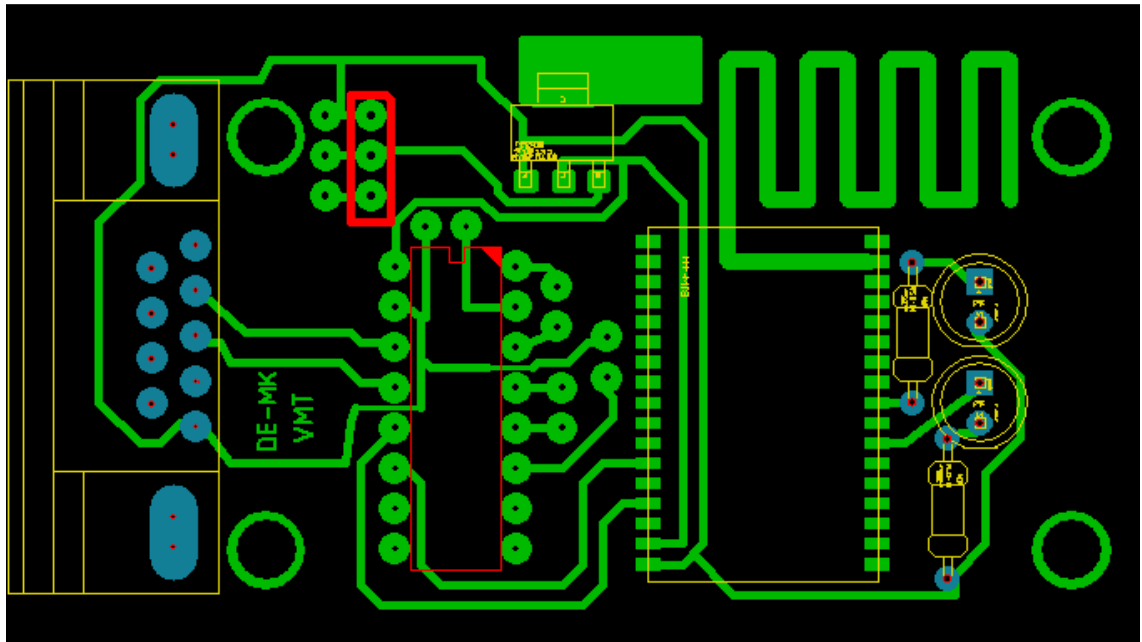
A soros port sebességét az „ATL1” paranccsal 9600 bps-ra állíthatjuk be, „ATR0”-val mester módba állíthatjuk, „ATO1” paranccsal pedig kikapcsolhatjuk az automatikus csatlakozást.

Az „ATF” paranccsal indíthatjuk el a Bluetooth modul kereső funkcióját, amely szövegesen kilistázza a talált eszközök Bluetooth nevét és címét. Miután már ismert az eszköz Bluetooth címe, csatlakozni is tudunk hozzá. A modulnak az „ATD=...” paranccsal kiválasztjuk, hogy melyik eszközhöz szeretnénk csatlakozni. Ezt az eszközcímet a modul elmenti és az „ATA” parancs kiadása után csatlakozik hozzá. A legtöbb Bluetooth eszköz használ PIN kódot. Ezért csatlakozás előtt beállítjuk az eszközön a kódot az „ATP=...” paranccsal. A kapcsolat létrejötte után, a modul LINK lábára kapcsolt LED folyamatosan világít, ami a kommunikáció működését jelzi. Mivel a BTM-112-es modul csak az SPP protokollt támogatja, olyan eszközökkel képes kommunikálni, amelyek szintén támogatják ezt a protokollt.

A Bluetooth egyik nagy előnye, hogy elfedi az eszközök közötti különbséget, azaz platformfüggetlen. Ez abból látszik, hogy a mikrovezérlő és a BTM-112 modul 9600 bps-os sebességgel, míg a vevő és a robot 19200 bps-al kommunikál. Az Bluetooth modulok puffereklik az adatot, így a sebességkülönbség miatt nincs információvesztés.

3.4. Bluetooth – RS232 átalakító

A roboton található soros port az RS232 szabványnak megfelelő jelszinteket használ (logikai 0: 3V és 15V között; logikai 1: -15V és -3V között [9]). Ugyanezt a jelszintet alkalmazzák a számítógépek soros portjain is. Ennek köszönhetően a robot közvetlenül rákapcsolható a PC soros portjára. Az ismertetett Bluetooth modul azonban nem ezt a jelszintet használja (logikai 0: 0V és 0,8V között; logikai 1: 2,4V és 3,3V között), így jelszint átalakításra van szükség. Szükséges továbbá az áramkör tápellátásának biztosítása, ugyanis a soros port nem gondoskodik energiaellátásról. A Bluetooth modul működéséhez szükséges tápfeszültség: 3V-3,6V. A roboton található ultrahangos távolságmérő szenzor 5V tápfeszültséggel működik. Erről 3,3V-os stabilizátor IC-vel levehető az áramkör táplálásához szükséges feszültség. Jelszint átalakításról a Maxim Semiconductors által gyártott IC, a MAX3232 gondoskodik. Az átalakító beültetési rajza a 6. ábrán látható.



6. ábra: Bluetooth - RS232 átalakító

A kapcsolás 3,3V-ról működtethető, kevés alkatrészt igényel és kétirányú kommunikációt biztosít. Az áramkör RS232 jelszintű csatlakozójára kapcsolódik a robot, a TTL jelszint oldalára pedig a Bluetooth modul. Mivel az egész áramkör tápellátása 3,3V-ról történik a TTL oldal TX lábán a logikai 1-nek megfelelő feszültségérték nem haladja meg a 3,3V-os értéket. Ezáltal a Bluetooth modul és az átalakító között nincs szükség feszültségosztó ellenállások alkalmazására.

A Bluetooth modul Link kimeneti lábára kapcsolt LED jelzi a kapcsolat állapotát. Slave üzemmódban ez a LED folyamatosan villog, ha nincs Bluetooth kapcsolat. Ekkor a modul vezérelhető AT parancsokkal. Csatlakozás után a LED folyamatosan világít, ekkor a modul adat módban van. Ez azt jelenti, hogy az RX lábra érkező adatot Bluetooth-on továbbítja a hozzá csatlakoztatott eszközhöz, mintha a robot és a távirányító soros vezetéken össze lenne kötve. Ha a kapcsolat megszakad, a modul újra AT parancs módba vált, a LED villog. Lehetőség van AT parancs módba visszaugrani a Bluetooth kapcsolat bontása nélkül, amennyiben a modul „escape sequence” funkciója aktiválva van (ATX1 paranccsal aktiválható) és a modul a „+++” karaktersorozatot kapja 1000 ms-os időhatáron belül. A data lábra kapcsolt LED az adatforgalmat jelzi. Ez akkor világít, ha éppen adatcserét végez a modul.

Ezzel az áramkörrel a robot képes Bluetooth kapcsolatot létesíteni a távirányító eszközökkel. A modul beállításait csak egyszer kell alkalmazni, ezeket az eszköz tárolja.

4. Mikrovezérlős távirányító áramkör

Ez a távirányító két egységből épül fel, lehetővé téve a kétkezes irányítást. Segítségükkel több üzemmódban is irányíthatjuk a robotot. Ezekről a robotban futó program ismertetésekor lesz szó.



7. ábra: Mikrovezérlős távirányító

4.1. Adatfolyam iránya

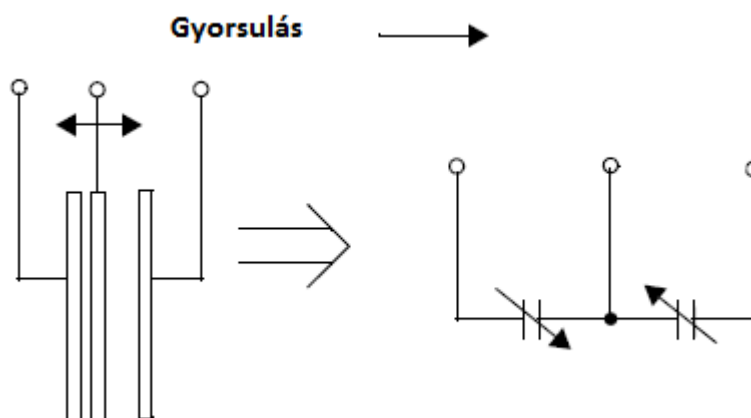
Mivel a robothoz csatlakoztatott Bluetooth adapter csak egy eszközhöz képes csatlakozni egyszerre, nem tud közvetlenül mindkét áramkörtől adatot fogadni. Ezért Bluetooth kapcsolatot csak az egyik áramkör létesít a robottal. Ez fogja továbbítani a másik áramkör által szolgáltatott adatokat is. Annak érdekében, hogy a két áramkör mozgatója szabadon történjen, és ne akadályozzák vezetékek, a kommunikáció 868 MHz-es rádiómodulok segítségével jön létre. Miután a robottal Bluetooth kapcsolatban lévő áramkör (továbbiakban A áramkör) megkapta a másik áramkör (továbbiakban B áramkör) által mért adatokat, azok mellé társítva saját mért adatait, csomag formájában továbbítja azokat a robot felé.

4.2. Gyorsulásmérő modul



8. ábra: MMA7361 modul

A felhasznált gyorsulásmérő áramkör egy MMA7361 típusú gyorsulásmérő IC-t tartalmaz [10]. Az IC egy kapacitív érzékelőcellából (g-cella) és egy jelformáló IC-ből áll egy tokozásban. A g-cella poliszilikon anyagból készült. Működése modellezhető: két álló kondenzátorlemez, melyek között egy mozgó lemez található a 9. ábra szerint.



9. ábra: g-cella modell [10]

Ez felfogható két szabályozható kapacitású kondenzátorként. Gyorsulás hatására a középső lemez elmozdul, az egyik lemezhez közeledik, a másiktól távolodik. Ennek hatására változik a két kondenzátor kapacitása a $C = \varepsilon \cdot \frac{A}{d}$ képlet szerint, ahol ε dielektromos állandó, A lemezek felülete és d lemezek távolsága. A jelformáló IC a kondenzátorok kapacitásváltozásával arányos analóg kimeneti feszültséget szolgáltat. Ebből lehet következtetni a lemezekre merőleges tengely irányába eső gyorsulásra.

A gyorsulásmérő IC három, egymásra merőleges irányba képes gyorsulás meghatározására. Két érzékenységi érték közül választhatunk az alkalmazási területnek megfelelően: 1,5g vagy 6g. Rendelkezik továbbá egy „0g detect” nevű

kimenettel. Erre a kimentre az IC egy impulzust ad, ha mindhárom tengely mentén 0g gyorsulást érzékel. Ez használható szabadesés érzékelésére.

Az IC tápellátása $V_{cc}=2,2-3,6V$ feszültséggel történik. A kimeneteken mért analóg feszültség a tápfeszültséggel arányosan változik. 0g gyorsulásnak $V_{cc}/2$ feszültség felel meg. A kimenetek a mikrovezérlő analóg-digitál átalakítójára csatlakoznak. A gyorsulásmérő kimenetei és a föld közé 3,3 pF-os kondenzátorok vannak bekötve. Ezek szerepe a kimeneti egyenfeszültségre rátevődő zaj szűrése, amely az IC belső 11 kHz-es oszcillátorából származik.

A mérési hibák megelőzése érdekében a tápfeszültséget fix értéken kell tartani. Ha a tápfeszültség változik, ugyanahhoz a gyorsulásértékhez más kimeneti analóg feszültségérték társul. Ennek eredménye a kézmozgás tartományának változása (például: a robot a kéz kisebb mértékben történő elfordítására már elindul egy bizonyos irányba). Az áramkör táplálása 4 db AA elemről/akkumulátorról történik. A telep merülésével csökken a kapocsfeszültség, ami épp az előbb említett hibát okozná. Mindkét áramkör tartalmaz feszültség stabilizátor IC-t, amely az áramkör bemenetére konstans 3,3V feszültséget szolgáltat. Amennyiben a telep feszültsége oly mértékben lecsökkenne, hogy a stabilizátor kimeneti feszültsége 3,3V alá csökken, a mikrovezérlő leállítja az áramkört és jelzi, hogy a telep lemerült.

4.3. Rádiófrekvenciás modulok

A mikrovezérlős áramkör két egységből áll. Hogy a két áramkör mozgatása szabadabb legyen, fizikailag függetlenek kell legyenek egymástól. A két áramkör külön tápról működik, valamint a kommunikáció is vezeték nélkül történik. Közöttük egyirányú kommunikáció jön létre: a B áramkör küldi a mért adatokat az A-nak. Mivel nincs szükség kétirányú kommunikációra, ezen feladat elvégzésére RF modul alkalmazható. Sebessége elmarad a Bluetooth-hoz képest, nem megoldott a zavarcsökkentés, viszont fogyasztása az alkalmazott Bluetooth modulhoz képest alacsonyabb. A zavarcsökkentés szoftveres módon megoldható, az adatátvitel sebessége pedig elegendő ebben az alkalmazásban.



10. ábra: HM-R868S és HM-T868S modulok

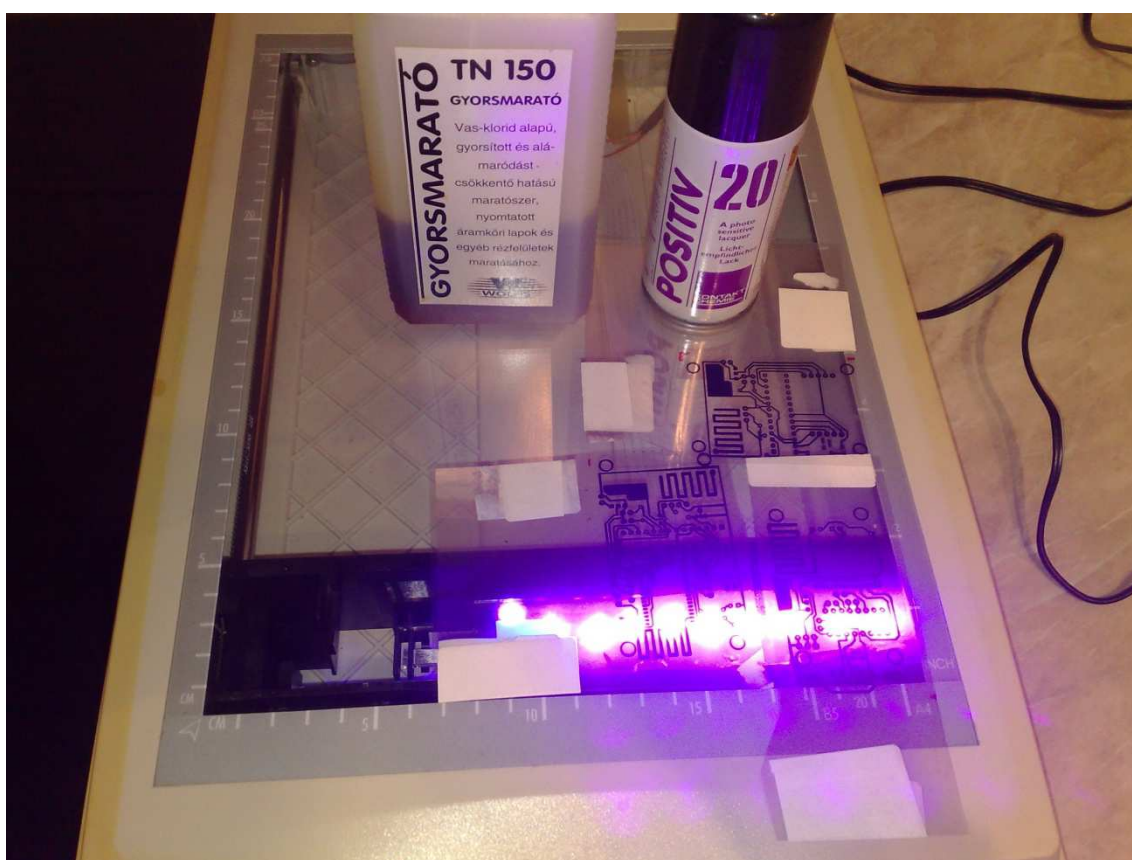
Az adó áramkör a 10. ábrán látható HM-T868S modul. Táplálásához 2,5-5V feszültség szükséges, áramfelvétele 31 mA. Az átviteli sebesség 600-9600 bps között választható. A két egység közötti távolság elérheti akár a 160 m-t is, ami jóval több, mint amire itt szükség van.

A vevő áramkör paraméterei hasonlóak az adóéhoz, ennek áramfelvétele kevesebb (10mA) valamint rendelkezik egy ENABLE bemenettel, amely a modul engedélyezését illetve készenlétbe helyezését állítja. A vevő modul ezen bemenetét az áramkör bekapcsolásakor 0 logikai állapotra kell kapcsolni, majd a modul inicializálása után 1 állapotba.

A modulok nem rendelkeznek beépített antennával. A legnagyobb hatótáv elérése érdekében az antenna hossza a rádiófrekvencia hullámhosszának $\frac{1}{4}$ -e. Ezek a modulok 867,84 MHz-en kommunikálnak, vagyis az ehhez tartozó hullámhossz: $\lambda = c/f$ [11], ahol $c = 3 \cdot 10^8 \text{ m/s}^2$, vagyis $\lambda = 0,34 \text{ m}$. Így az alkalmazott antennaméret ezen érték $\frac{1}{4}$ -e, vagyis 8,6 cm. A helymegtakarítás végett az alkalmazott antenna mérete ennél kisebb. Ebben az esetben ez nem jelent gondot, mert a szükséges hatótáv még az 1m-t sem haladja meg.

4.4. Az áramkör megvalósítása

Az áramkörök tervezése számítógépen, az Abacom szoftverével, Sprint Layout 5.0-ás verziójával készült. A program könyvtára rengeteg alkatrészt tartalmaz, többek között az áramkörben használt mikrovezérlőt is. Beállítható a vezetékek vastagsága valamint a forrszemek mérete és a furat átmérője. Az alkatrészek rajza több rétegre, úgynevezett layer-ekre vannak felosztva. Egyes rétegek tartalmazzák a forrszemeket, mások pedig a forrszemhez tartozó feliratokat (láb kiosztást), az áramköri elemek alakját. Nyomtatáskor természetesen csak a forrszemeket és vezetékeket tartalmazó rétegeket kell kinyomtatni. A többi réteg a tervezésben nyújt segítséget.



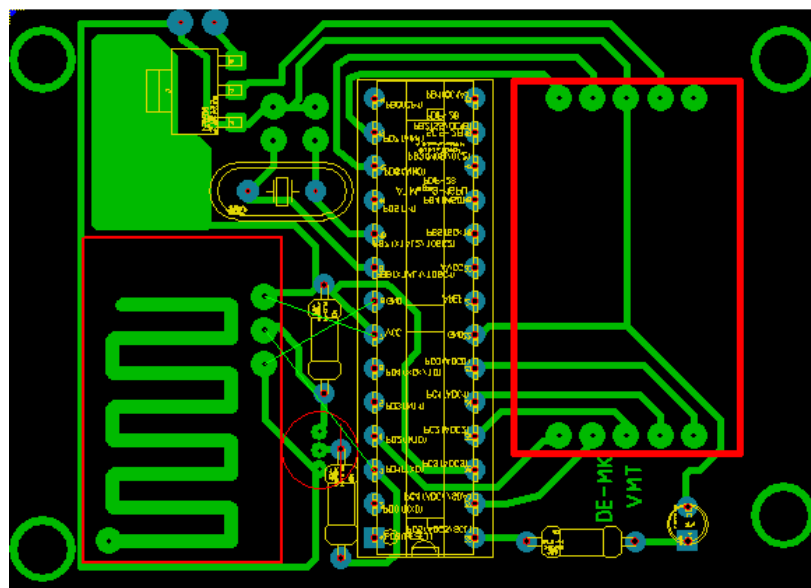
11. ábra: Áramkörök megvalósítása

Az áramkörök fizikai megvalósítása fotokémiai eljárással történt. Az elkészített beültetési rajz fóliára lett kinyomtatva. A nyáklemez a 11. ábrán látható Positiv20-as, UV fényre érzékeny lakkal lett befújva. Ezután a lemez sütőben, 60°C-os hőmérsékleten 15 percig száradt. A megszáradt lemezre került a kinyomtatott áramkör rajza a fólián. A képen látható, házi készítésű UV szkennel segítségével történt az áramkör átvilágítása a lemezre. A szkennel kocsija nagy fényerejű UV LED-eket mozgat.

Ahol az UV fény éri a lakkot, megváltozik a lakk szerkezete. Levilágítás után a panel NaOH oldatba került, ami leoldja a lakk azon részét, amelyet UV fény ért. A kitakart részeken megmarad a lakkréteg. Szintén a képen látható TN 150-es gyorsmarató leoldja a panelről azt a rézréteget, amelyet nem fed le a lakk. Így végeredményben a panelon kirajzolódik az áramkör képe. Következő lépésként vasfűrészlappal történt a lemezek méretre vágása, majd 1mm-es fúróval a furatok elkészítése. Utolsó lépésként pedig az alkatrészek elhelyezése és a forrasztás történt.

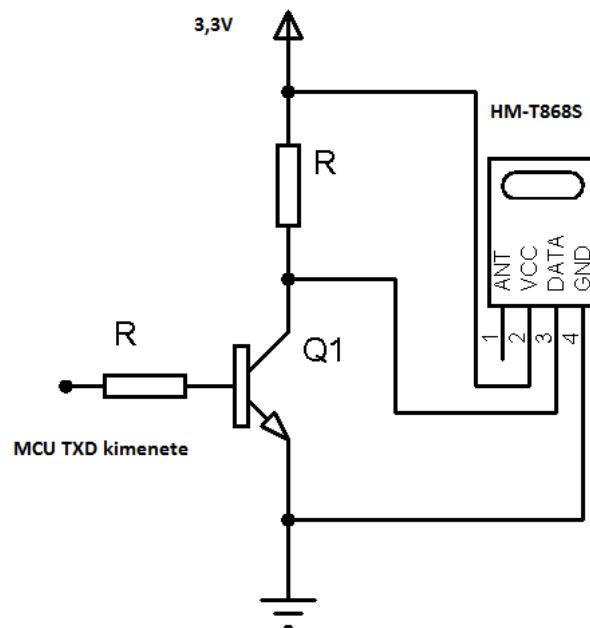
Mindkét adó áramkör az ATmega8-as mikrovezérlőre épül[12]. Táplálása 3,3V-ról történik, ami az adatlapban szereplő minimális feszültségértéktől kevesebb, azonban gyakorlati tapasztalat szerint minden periféria és a processzor is hibátlanul működik ezen a feszültségen. Erre azért van szükség, mert az áramkörök elemekkel ill. akkumulátorokkal működnek. Ezáltal csökken a fogyasztás, valamint az áramkör tovább működik, mintha 5V-ról táplálnánk. Mivel egyes alkatrészek maximálisan 3,3V-ról működtethetők (Bluetooth modul), nincs szükség szintillesztésre, vagyis kevesebb alkatrészre van szükség. Egyedül a mikrovezérlők programozásához kell elérni legalább a katalógusban szereplő 4,5V-ot. Ezért a mikrovezérlők felprogramozása az áramkörből kiemelve, próbapanelon történik. Későbbi újraprogramozhatóság érdekében a mikrovezérlők a panelon IC foglalatba csatlakoznak, így forrasztás nélkül eltávolíthatóak. Az órajelet 16 MHz-es, külső kvarc kristály szolgáltatja.

A 12. ábrán látható a B áramkör beültetési rajza.



12. ábra: B áramkör beültetési rajza

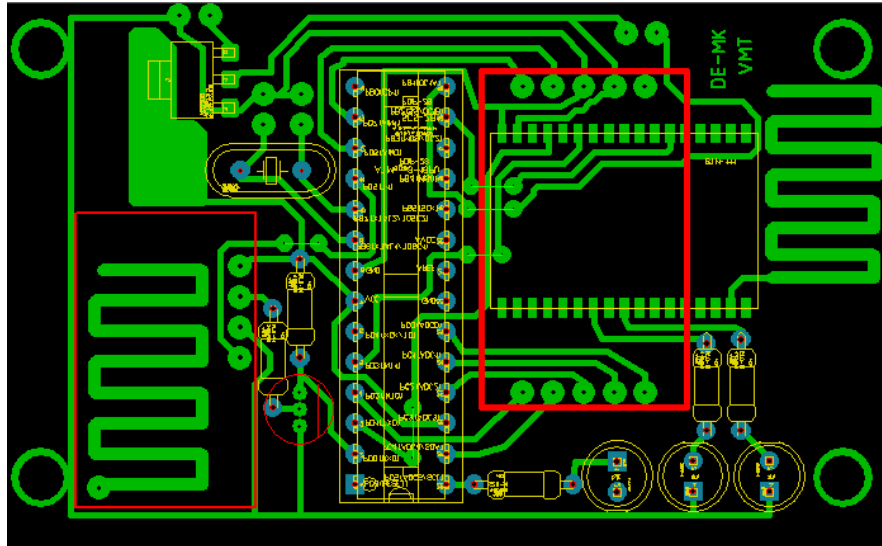
A stabil 3,3V feszültséget az LM 1117IMP-3.3 típusú stabilizátor IC szolgáltatja. Az IC 800mA-ig terhelhető, 1,2V feszültség esik rajta. Amíg az elem 4,5V-nál magasabb feszültséget biztosít, a kimeneti feszültség stabil, nem csökken. Az IC kimenete táplálja az összes áramköri elemet. A gyorsulásmérő modul X, Y és Z kimenetein analóg feszültség jelenik meg. Ezek a mikrovezérlő analóg-digitál átalakítójához csatlakoznak (ADC0-ADC2 csatornák). A modul alvó állapotba is helyezhető, a SLEEP kivezetésen keresztül, amely a mikrovezérlő PC4-es lábára csatlakozik. A 0g kimenet a szabadesés érzékelését teszi lehetővé, ez a mikrovezérlő INTO lábára csatlakozik, amely külső megszakítást generáló bemenet. A modul GS bemenete a mikrovezérlő PD6-os lábára van kötve. Ennek segítségével a modul érzékenységét lehet szabályozni. A rádiófrekvenciás modul bemenete a mikrovezérlő TXD lábára csatlakozik. Mivel soros kommunikáció esetén a készenléti állapot a kimenet logikai 1-re állítását jelenti a mikrovezérlőnél, ezért egy egyszerű inverter van beiktatva az RF modul DATA és a mikrovezérlő TXD lába közé a 13. ábrán látható kapcsolás szerint. Így a modul bemenete készenlétkben a földre van húzva.



13. ábra: Inverter rádiómodulhoz

A mikrovezérlő PC5 lábára egy LED csatlakozik. Bekapcsolt állapotban ez világít, valamint villogással jelzi, ha a telep lemerült.

Az A áramkör beültetési rajza a 14. ábrán látható.



14. ábra: A áramkör beültetési rajza

A kapcsolat csupán annyiban tér el a B áramkörétől, hogy ezen még egy Bluetooth modul is megtalálható. Itt az RF modul vevőegysége található meg. Ennek kimenete szintén egy inverterre csatlakozik, amely a mikrovezérlő RXD lábára van kötve. Megtalálható az RF modulon egy ENABLE bemenet is, amely a modul be/kikapcsolását teszi lehetővé. Ezt a mikrovezérlő PD5-ös kimenete kapcsolja. A Bluetooth modul RX bemenete a mikrovezérlő TXD kimenetére csatlakozik. A modul TX kimenete pedig a mikrovezérlő INT1 külső megszakítást generáló lábára van kötve. Ennek oka, hogy a mikrovezérlő hardveres RXD bemenetét az RF modul használja. Mivel ebben az alkalmazásban a kommunikáció a Bluetooth-on keresztül is csak a távirányítótól a robot felé történik, ezt a vezetéket nem használjuk. Későbbi fejlesztés céljából azonban fizikailag mégis be van kötve. Így létrehozható egy megszakításon alapuló szoftveres soros bemenet. A Bluetooth RESET bemenete a mikrovezérlő PB2 lábára van kötve. Ezen keresztül lehet a modult alaphelyzetbe állítani. A modul vezérel még két LED-et, amelyek a LINK és DATA kimenetére vannak kötve. A LINK-re kapcsolt LED akkor világít, ha a modul csatlakoztatva van a vevővel, a DATA LED-je pedig az adatforgalmat jelzi.

4.5. Az áramkör működése

Bekapcsolás után a B áramkör az RF modul adón keresztül folyamatosan küldi ki az adatokat, amelyeket a gyorsulásmérő kimenetein mér. Az adatok csomag formájába vannak rendezve: 7 bájtból épül fel egy csomag, ahol az első bájt a 'J' karakter kódja. Ezt követi kétszer az 'A' karakter kódja, majd a három tengely irányában mért gyorsulás értéke. A csomagot a 'Z' karakter kódja zárja. Az A áramkör bekapcsolás után csatlakozik Bluetooth-on a robothoz, majd várja a B áramkör felől érkező csomagot. Amint a csomag megérkezett, azokból kinyeri a mért adatokat, hozzátársítja saját mért adatait, majd újabb 7 bájtos csomagot formál: az első bájt értéke 0 és a következő 6 bájt a mért adatok értékei.

Mivel a két áramkör többnyire ugyanazokat a perifériákat használja, ezek együtt kerülnek bemutatásra.

4.6. Analóg-digitál konverter

Az ATmega8-as mikrovezérlő 10 bites analóg-digitál átalakítóval rendelkezik, 6 bemeneti csatornával. Ezt a perifériát használjuk a telep feszültség mérésénél és a gyorsulásmérő kimeneti analóg feszültségének digitalizálásánál.

Inicializálás

Az analóg-digitál konverter beállításait regiszterein keresztül adhatjuk meg. A 15. ábrán látható az ADMUX és ADCSRA regiszter:

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

15. ábra: ADCSRA és ADMUX regiszter [12]

Az ADCSRA regiszter ADEN bitjét 1-esre állítva engedélyezzük a konvertert. Az alsó három bit a jel mintavételezésének a frekvenciáját határozza meg (előosztó). Ezek értékeit az adatlap szerint kell kiválasztani. A mintavételezési frekvencia úgy keletkezik, hogy a processzor órajelét elosztjuk az előosztóval beállított értékkel. Az előosztó

értékét úgy kell megválasztani, hogy a mintavételezési frekvencia 50 és 200 kHz közé essen [12]. Ezen intervallumon belül, nagyobb frekvencia esetén az átalakítás gyorsabb, alacsonyabb frekvencia esetén viszont pontosabb. Ebben az esetben mindhárom bit értéke 1, vagyis a processzor órajelét (16 MHz) 128-al kell elosztani. Tehát a mintavételezési frekvencia 125 kHz lesz.

Az átalakítónak szüksége van egy referencia feszültségre. Ez lesz az a feszültség, amelyet ha a bemenet elér, az átalakítás végén a legnagyobb digitális értéket kapjuk (10 biten ez 1023). A REFS1 és REFS0 biteket 1-esre állítva, az adatlap szerint a referencia feszültség a mikrovezérlőben előállított stabil 2,56V lesz.

A mérési eredményeket az ADC nevű 10 bites regiszterből olvashatjuk ki. Ez a regiszter tulajdonképpen két regiszterből áll. Ennek oka, hogy a mikrovezérlő 8 bites rendszerű, így minden regisztere legfeljebb 8 bitből áll: ADCH és ADCL. A C fordítónak köszönhetően a forráskódban ADC néven közvetlenül elérhetjük ezt a regisztert, nem kell külön két regisztert kiolvasni, majd bitműveletekkel összefűzni 10 bites számmá.

Az ADMUX regiszter ADLAR bitje állítja be a 10 bit elrendezkedését az ADCH és ADCL regiszterekben. Ha ez a bit 0, az eredmény alsó 8 bitje az ADCL regiszterben található meg, a felső két bit pedig az ADCH regiszterben. Ellenkező esetben, ha az ADLAR bit 1-es, az eredmény felső 8 bitje az ADCH regiszterben, az alsó 2 bit pedig az ADCL regiszterben helyezkedik el. A kódban ez a bit 1-esre van állítva. Mivel az eredmény alsó két bitje zajokat tartalmaz, ezeket elhagyva az eredmény pontosabb. Ezért elég csak az ADCH regisztert kiolvasni. Ennek eredménye, hogy a kapott érték 0-255-ig terjed, amely kisebb felbontást eredményez, mint a legjobb elérhető, de ebben az alkalmazásban ez is elegendő.

4.7. ADC átalakítás

Az inicializálás után a konverter alapbeállításait megadtuk. A mérés az „ADC_measure(uint8_t ch)” függvény meghívásával indíthatjuk el. Ez a függvény paraméterként annak a csatornának a sorszámát várja, amelyre kapcsolt feszültséget digitalizálni szeretnénk. Az ADMUX regiszter MUX3-MUX0 bitjei választják ki ezt a csatornát. Az ADCSRA regiszter ADSC bitjét 1-esre billentve elindul az átalakítás. Az átalakítás elkészültével az ADCSRA regiszter ADIF bitje 1-esre vált. Amíg ez

megtörténik, a program vár, a mikrokontroller nem végez más feladatot. Az ADIF jelzőbit törlése után a függvény visszaadja az ADCH regiszterben tárolt eredményt.

4.8. Elem feszültségének mérése

A gyorsulásmérő moduloknál láthattuk, hogy a tápfeszültség csökkenése befolyásolja a kimeneten megjelenő analóg jel nagyságát, ugyanarra a pozícióra. A helyes működés érdekében figyelni kell arra, hogy az áramkör feszültségellátása stabil legyen. A 3,3V-os stabilizátor éppen ezt a feladatot látja el. Mivel a stabilizátoron is esik feszültség (adatlap szerint 1,2V), a 3,3V csak akkor jelenik meg a kimenetén, ha a telep feszültsége legalább 4,5V. Ha ez alá esik a telepfeszültség, a stabilizátor kimenetén is csökkenni kezd a 3,3V feszültség, amely már megváltoztatja a mérési értékeket. A mikrovezérlő ezért bekapcsoláskor, majd bizonyos időközönként megméri a stabilizátor kimeneti feszültségét. Ha ez egy előre beállított értéknél kisebb, lekapcsolja a gyorsulásmérőt és jelez a felhasználónak egy LED villogtatásával. A feszültségmérés az analóg-digitál konverter segítségével történik.

A konverter eredménye az alábbi képlet szerint áll elő: $ADC = (V_{in}/V_{ref}) * 255$. Mivel alaphelyzetben a referenciafeszültség a belső 2,56V-ra van beállítva (inicializáláskor megadott érték), a V_{in} pedig a bemeneti 3,3V, az ADC eredménye a képlet szerint mindig nagyobb 255-nél. Hogy mégis észre lehessen venni a feszültségváltozást, meg kell cserélni a referencia feszültséget és a bemeneti feszültséget. A konverzió elkezdése előtt referenciafeszültségnek a tápfeszültséget kell megadni (ADMUX regiszterben, REFS1 és REFS0 segítségével), bemeneti feszültségnek pedig a referencia feszültséget (ADMUX regiszterben MUX3-MUX0 bitekkel). Ez azzal jár, hogy a tápfeszültség csökkenésére az ADC értéke nőni fog. Ezt az értéket összehasonlítva egy megadott felső értékkel lehet figyelni, hogy a tápfeszültség az alsó küszöb alá csökkent-e. A referenciafeszültség megváltoztatása utáni első mérés eredményét (az adatlap javaslata szerint) a program elveti. Ez a mérés ugyanis nagyobb hibákat tartalmaz. A konverzió után a referenciafeszültség visszaállítódik 2,56V-ra. Mivel itt is referencia feszültség-váltás történt, újabb mérést kell végezni a hibás eredmények elkerülésére.

A program elején, az analóg-digitál konverter inicializálása után meghívódik a „check_bat()” függvény, amely az előbbi feladatokat elvégzi. A telep feszültsége

azonban működés közben is csökken. Hogy ne terheljük le a mikrovezérlőt, bizonyos időközönként meghívódik a „check_bat()” függvény. Ez a mikrovezérlő számláló perifériája segítségével történik.

4.9. Időzítő beállítása

Bit	7	6	5	4	3	2	1	0	
	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	–	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

16. ábra: TCCR1B és TIMSK regiszterek [12]

A program indításakor történik az időzítő inicializálása. A mikrovezérlőben 3 számláló is van, ezek közül az Timer/Counter 1-et használjuk. Ez a számláló 16 bites, vagyis 2^{16} értékig tud elszámolni. Inicializáláskor meg kell adni a számláló órajelét. Ezt a TCCR1B regiszter segítségével adhatjuk meg, a CS12-CS10 bitek segítségével. A kiválasztott órajel az ($F_{CPU}/1024$), azaz 15,6 kHz lesz. Egy léptetés a számlálóban tehát 6,4 us időközönként történik. Amint az órajel be lett állítva, a számláló azonnal indul. A léptetés automatikusan történik a háttérben, nem terheli a processzort, így több erőforrás marad más feladatokra. A TIMSK regiszter TOIE1 bitjével engedélyezhetjük a Timer1 túlszordulása esetén létrejövő megszakításkérélmét. Ez azt jelenti, hogy amint a számláló elérte a maximális értéket (16 biten ez 65 535), 0-ra vált és megszakítást generál. A főprogram futása leáll, meghívódik az „ISR(TIMER1_OVF_vect)” nevű függvény, majd végrehajtása után a főprogram folytatódik ott, ahol abbamaradt. Több „ISR()” függvény is lehet deklarálva a forráskódban, hiszen több periféria is generálhat megszakításkérélmét. Paraméterként ezeknek az ISR függvényeknek egy konstanst kell átadni, amelyből a mikrokontroller tudja, hogy melyik megszakításhoz melyik függvény tartozik. Ezeket a konstansokat az adatlap tartalmazza. Megszakítás csak akkor hajtódik végre, ha a program elején engedélyeztük az általános megszakítási kérélmeket a „sei()” függvény meghívásával. Számlálónk tehát $2^{16} \cdot 6,4 \mu s = 0,4$ másodpercenként csordul túl és hívja meg a hozzá tartozó ISR függvényt. Ez a függvény egy „timer” nevű változó értékét növeli eggyel

minden megszakítás után. Ha a változó értéke eléri a 100-at, az ISR függvény meghívja a „check_bat()” függvényt és a „timer” változót 0-ra állítja. Ebből látszik, hogy a telep feszültségmérése körülbelül 40 másodpercenként történik. Ez az időintervallum elegendő két mérés között, sőt még növelhető is, ugyanis a telep feszültsége nem változik hirtelen működés közben sem, ugyanis az áramkör áramfelvétele kicsi és a kommunikáció létrejötte után állandó értékű.

4.10. USART modul

A mikrovezérlő tartalmaz egy USART (Universal Synchronous/Asynchronous Receiver Transmitter) nevű perifériát, amelyet soros kommunikáció megvalósítására szolgál. Ezt használva a soros kommunikáció során nem kell időzítőkkal és megszakításokkal, a processzor erőforrásait elhasználva, szoftveres módon megoldani az adatcserét. A periféria beállítása után egyszerűen történik a kommunikáció a hardverelemek között. A perifériát használhatjuk szinkron és aszinkron módban. Ebben az alkalmazásban aszinkron módban alkalmazzuk. A B áramkör csak a periféria küldő részét használja (Tx), ez van összekötve az adó rádiómodullal. Az A áramkörben a periféria fogadó részére (Rx) csatlakozik a vevő rádiómodul, a küldő részére (Tx) pedig a Bluetooth modul. Mindkét áramkör hasonló beállításokat használ. Ezeket a periféria regiszterein keresztül adhatjuk meg.

4.10.1. USART beállítása

A periféria Tx kimenete a mikrovezérlő 3-as lábára van kivezetve, amely egyben a D port PD1 digitális kimenete is. Alaphelyzetben ez a csatlakozó bemenetként szolgál. A D port adatirány regiszterében (DDRD regiszter) be kell állítani, hogy a PD1 kimenetként szolgáljon.

Bit	7	6	5	4	3	2	1	0	
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	UCSRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

17. ábra: UCSRB regiszter [12]

Az UCSRB (Usart Control and Status Register B) regiszter TXEN és RXEN bitjeit 1-esre állítva aktiválhatjuk a perifériát. A B áramkör esetén csak a TXEN bitet állítjuk be, mert itt csak adatot küldünk.

Bit	7	6	5	4	3	2	1	0	
	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	UCSRC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	0	0	0	0	1	1	0	

18. ábra: UCSRC regiszter [12]

Az UCSRC (Usart Control and Status Register C) regiszterben beállíthatjuk az adatbitek számát. Az UCSZ1 és UCSZ0 biteket 1-re állítva a kommunikáció 8 adatbitet tartalmaz. Ez a regiszter és az UBRRH regiszter azonos címen helyezkedik el a regiszterek között. Ha az URSEL bitet 1-esre állítjuk, az adat az UCSRC regiszterbe kerül, ha 0-ra állítjuk, akkor az UBRRH regiszterbe. A soros kommunikáció elméleti részénél láthattuk, hogy az adatátvitel sebességét a másodpercenként átvitt bitek száma jelenti. Ezt a 16 bites UBRR (Usart Baud Rate Register) regiszterrel állíthatjuk be, amelyet az UBRRH és UBRL 8 bites regiszterek alkotnak. Az adatlap szerint, ha ismert az adatátvitel sebessége (baudrate, bps-ban megadva) és a mikrovezérlő órajele, kiszámítható az UBRR értéke az alábbi képlet szerint:

$$UBRR = \frac{F_{CPU}}{16 * baudrate} - 1 [12].$$

Az így kapott érték alsó 8 bitje kerül az UBRL regiszterbe, a felső 8 bitje pedig az UBRRH regiszterbe. Sikeres kommunikációhoz elengedhetetlen, hogy a kommunikáló eszközök adatátviteli sebessége azonos legyen.

Mindezen beállításokat elvégezve készen állunk az adatok fogadására illetve küldésére. A periféria UDR (Usart Data Register) nevű regiszterét használjuk fel az adatok küldésére és fogadására. Ha adatot szeretnénk küldeni, ellenőriznünk kell hogy a periféria éppen szabad, vagyis nem küld és nem fogad adatot. Ezt az UCSRA regiszter UDRE (Usart Data Register Empty) bitjének vizsgálatával végezhetjük el. Ha ez a bit 1, az elküldeni kívánt adatot beleírhatjuk az UDR regiszterbe. Ekkor az UDRE bit 0-ra vált. Addig marad 0, amíg az adatküldés folyamatban van. Újabb bájtot csak akkor írhatunk az UDR regiszterbe, miután az UDRE újból 1-esre vált. Ha adatot akarunk fogadni, az UCSRA regiszter RXC (RX Complete) bitjét kell vizsgálni. Addig kell várni, amíg ez a bit 1-esre vált. Ha ez megtörtént, az UDR regiszterből kiolvashatjuk a fogadott bájtot.

5. Mobiltelefonos távirányító

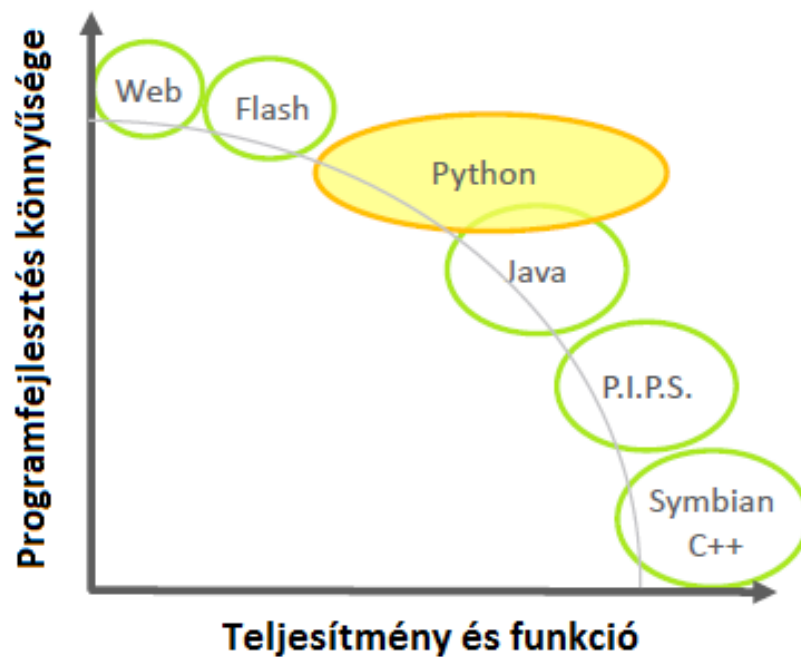
A robot mozgatása megvalósítható bármilyen eszközzel, amely rendelkezik Bluetooth adapterrel és gyorsulásmérő áramkörrel. Mivel a mai mobiltelefonok többsége eleget tesz ezeknek a feltételeknek és lehetővé teszik a felhasználó által írt programok futtatását, a robotot vezérelhetjük ezek segítségével is. Az eszközöknek csupán megfelelő formátumban kell kiküldeni az adatokat. A mikrovezérlős távirányító esetében egy adatcsomag 7 bájtból áll. Az első bájt azonosítja a távirányítót, további 6 bájt pedig tartalmazza a két áramkör által mért nyers adatokat. A mobiltelefonok csak egy gyorsulásmérővel rendelkeznek, így ebben az esetben egy adatcsomag 4 bájtból áll. A robotvezérlő program két elterjedt mobiltelefonos operációs rendszer alá készült el: Symbian-ra és Androidra. A tengelyek mentén mért gyorsulások értékeit a két operációs rendszer eltérő formátumban és tartományban ábrázolja. Ahhoz, hogy ugyanolyan pozícióhoz ugyanazok a számadatok tartozzanak, matematikai átalakítások szükségesek. Hogy ne terhelődjön a mobilkészülék ezen műveletek elvégzésével, a robotban futó programot kell felkészíteni a különböző adatformátumok fogadására. A csomag első bájtja mondja meg a robot programjának, hogy milyen típusú operációs rendszertől érkezik az adat. Ha az első bájt értéke 1, a készülék Symbian, ha az érték 2, a készülék Android operációs rendszerű. A robot programja a kapott adatokat megfelelő átalakítások után feldolgozza.

5.1. Symbian operációs rendszerre írt vezérlő

Az robot vezérlését egy Nokia gyártmányú N95, Symbian operációs rendszerű, S60-as platformmal rendelkező mobiltelefonnal végeztük el.



19. ábra: Symbian operációs rendszeren használható programnyelvek [13]



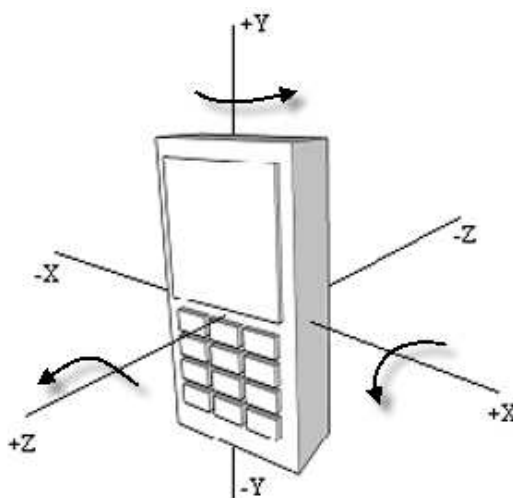
20. ábra: Hatékonyság és teljesítmény viszonyok [13]

A Symbian operációs rendszer egyfelhasználós, grafikus, többfeladatos rendszer. Előnye az operációs rendszerrel nem rendelkező telefonokkal szemben, hogy fejleszthető szoftverekkel. A Symbian a Nokia által fejlesztett keretrendszerrel rendelkezik. A rendszert C++ nyelven írták meg, ezáltal a leghatékonyabb programok is ebben a nyelven írhatók. Ezen felül futtathatók rajta többek közt Java, valamint Python nyelven íródott programok is. A Symbian C++ nyelv az ANSI C++ -hoz képest nagyon eltérő szerkezetű. Szigorú névadási szabályok jellemzik, hiányzik a kivételkezelés, helyette egy nehézkesen érthető ún. „leave”-elést megvalósító függvényeket kell létrehozni, a dinamikus memória-felszabadítás hiányzik (garbage collector), helyette a felhasználóra hárul a már nem használt memóriaterületek felszabadítása (stack pointer segítségével). A memória felszabadítás problémája nagyon fontos az ilyen, viszonylag kevés memóriával rendelkező eszközök esetében, hisz egy mobiltelefon sokkal ritkábban van újraindítva, mint egy asztali számítógép. Ezért egy rosszul megírt program nagyon lelassíthatja az eszközt [14]. Mindezekből következik, hogy a Symbian C++ nyelven történő programozás gyakorlott és tapasztalt programozókat kíván.

A 20. ábrán láthatóak a Symbianban használható programnyelvek teljesítmény és nehézségi fok alapján csoportosítva. Ez mutatja, hogy a Symbian C++ a leghatékonyabb és egyben a legnehezebben elsajátítható nyelv. Ehhez képest a Python teljesítmény szempontjából elmarad, azonban tanulhatósága és a fejlesztés gyorsasága jelentősen nagyobb.

5.2. Miért Python?

A Python egy objektumorientált, script alapú, kivételkezelést támogató, dinamikus adattípusokat használó, hatalmas könyvtárkészlettel rendelkező, egyszerű szintaktikájú nyelv [15]. A Symbian-os, S60-as szériákon használható script értelmező alkalmazást PyS60-nak nevezik. A programot egy egyszerű szöveges szerkesztőben lehet megírni, amit „.py” kiterjesztéssel kell elmenteni, majd a telefonra feltelepített PyS60-as alkalmazással lehet futtatni [16].



21. ábra: Mobilkészülék gyorsulás tengelyei

A programon belül valós időben, a telefon billentyűzetét használva is lehet interaktívan programozni illetve az előre megírt script-fájlokat futtatni. Egyik nagy hátránya, hogy a szintaktikai hibák csak futási időben derülnek ki, hisz a program soronként értelmezi, fordítja és hajtja végre az utasításokat. További gondot okoz még az is, hogy a script futtatásához előre telepíteni kell az értelmezőt és a hozzá tartozó kiegészítő könyvtárakat.

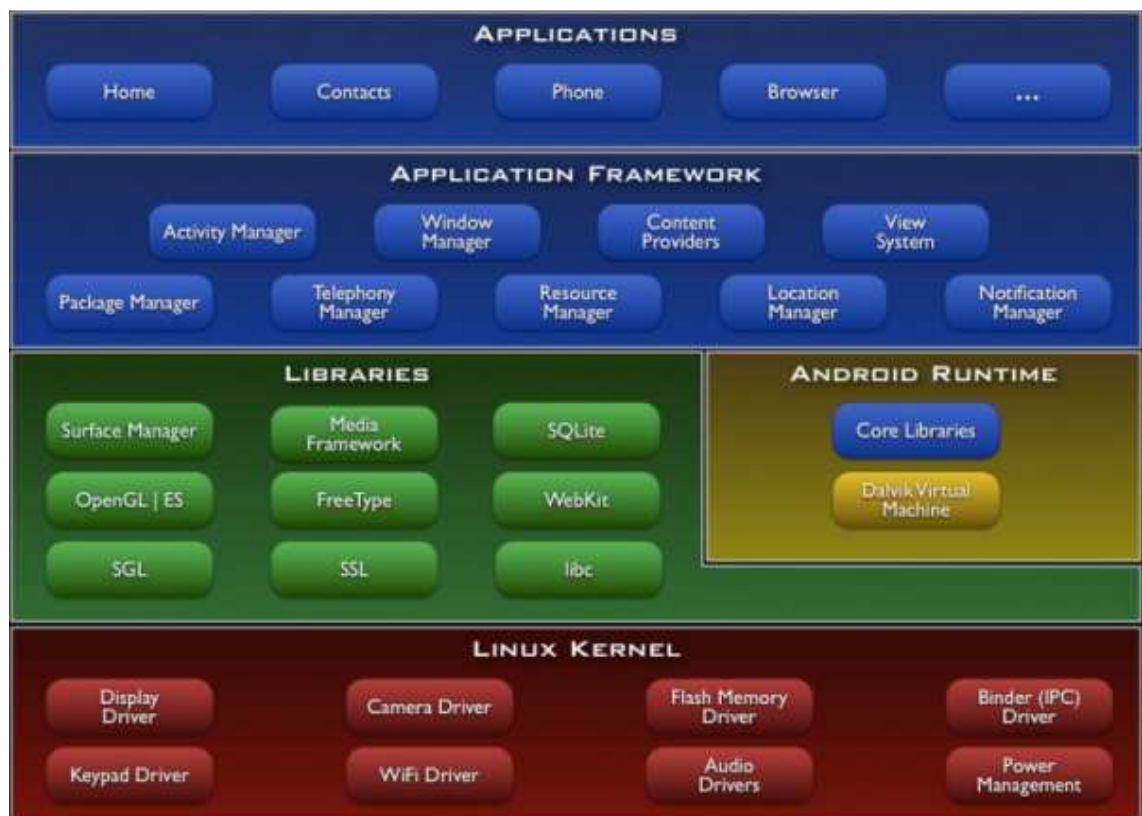
5.3. A program működése:

A script képes keresni, csatlakozni és kommunikálni különböző eszközökkel Bluetooth-on keresztül. Működése hasonló a mikrovezérlős áramköréhez: csatlakozik a robothoz, majd a beépített gyorsulásmérő áramkör adatait továbbítja a robotnak. Itt is csomagszerkezetben történik a kommunikáció. Az első küldött bájt értéke 1, amiből a robot tudja, hogy ez a Symbian-os mobiltelefonos távirányító. Ennek megfelelően csak további három bájtot vár. Az X és Y tengelyek adatai alapján a robot sebessége és mozgásiránya számítható ki. A Z tengely helyzetétől függően, ha a kijelző a $\bar{g}$ irányába

néz (fejjel lefelé tartjuk a telefont) a szervó motor helyzetét tudjuk változtatni az Y tengelyen mért adatok segítségével.

5.4. Android operációs rendszerre írt vezérlő

Az Android egy Linux kernel fölött futó, mobil operációs rendszer. A fejlesztők Java nyelven írhatnak alá programokat, az eszközt a Google által fejlesztett Java programkönyvtárakon keresztül vezérelve. A Linux kernel fölé egy virtuális gép került, amely a felhasználói felület kezeléséért és az alkalmazások futtatásáért felelős [17].



22. ábra: Android operációs rendszer felépítése [17]

A platform alapját a vörös színnel jelölt Linux kernel adja, amely tartalmazza a hardver által kezelendő eszközök meghajtó programjait. Ezt a készüléket gyártó cég készíti el. A kernel szolgáltatásait használják a Linux rendszerekben meglévő különféle programkönyvtárak, mint a libc, az SSL vagy az SQLite. Részben ezekre épül a Dalvik virtuális gép, amely egyáltalán nem kompatibilis a Sun (Oracle) virtuális gépével. A Java programok nem a Java-nál megszokott „.class” állományba kerülnek fordítás után, hanem egy nagyobb Dalvik Executable formátumba, amelynek kiterjesztése „.dex” és általában kisebb, mint a forrásul szolgáló .class állományok mérete. A virtuális gép más,

mint a Java alatti megszokott virtuális gép, vagyis a Java csak mint nyelv jelenik meg. A kék színnel jelölt részekben már csak Java forrást találunk, amelyet a virtuális gép futtat [17].

A robotvezérlő program Az Eclipse nevű szoftverfejlesztő környezetben(SDK) készült el. A programok fordítását az Android SDK végzi, amely tartalmaz egy számítógépen futtatható virtuális készüléket Android rendszerrel. Ennek segítségével tesztelhetők a programok. A kapcsolatot az Eclipse SDK és az Android SDK között az Eclipse-be beépülő Android ADT modul biztosítja.

5.5. Android program fejlesztése

A Symbian-os programtól eltérően, a végeredmény egy önmagában futni képes alkalmazás lesz, amelyet bármilyen Android operációs rendszerű készüléken futtathatunk más, előre telepített alkalmazás használata nélkül. Egy Android projekt több fájlból és mappából áll. Az „src” mappa tartalmazza az általunk megírt java forrásfájlokat. A „gen” mappa tartalmazza az automatikusan generált Java fájlokat. Az „Android 2.1-update1” mappa az ezen verziójú operációs rendszerhez szükséges fájlokat tartalmazza. Ez a mappa annak függvényében változik, hogy a projekt létrehozásakor melyik verziójú Android rendszert választjuk ki. Ennek az a jelentősége, hogy az alkalmazás a kiválasztottnál korábbi verziójú operációs rendszeren nem fog futni. A „res” mappa tartalmazza az alkalmazás külső erőforrásait. Ezek azok az elemek, amelyeket a programozó ad hozzá az alkalmazáshoz és az Android rendszer nem tartalmaz. Ebben található meg az alkalmazás ikonja, az alkalmazás ablakának leírása, a gombok és címkék feliratai. Az alkalmazás szempontjából nagyon fontos fájl az „AndroidManifest.xml” Ez a fájl írja le az alkalmazást. Tartalmazza többek között az alkalmazás nevét, verzióját, a futásához szükséges operációs rendszer legkisebb verziószámát és hogy melyik osztály indul el az alkalmazás megnyitásakor. Tartalmazza továbbá az alkalmazás engedélyeit. Ezek az engedélyek határozzák meg, hogy a program milyen erőforrásokhoz férhet hozzá a rendszerben. Telepítéskor a felhasználónak el kell fogadnia ezeket az engedélyeket, hogy később a program hozzáférhessen ezekhez az erőforrásokhoz. A 23. ábrán látható az alkalmazás indítása után. Az Androidban egy alkalmazás képernyőjét „activity”-nek nevezik [18].

A program több activity-ből állhat, amelyek egymástól függetlenek. Ez a program csupán ezt az activity-t tartalmazza, melynek neve „DaNImain”.



23. ábra: Robotvezérlő program Androidra

Az „AndroidManifest.xml” fájl ezt az activity-t állítja be alapértelmezettnek, vagyis ez fog elindulni az alkalmazás indításakor. Az activity grafikus felületét a „res\layout” mappában található „main.xml” fájl írja le. A kijelzőn megjelenik egy szövegdoboz és három gomb. Az xml fájl írja le ezek tulajdonságait és helyzetét. Hozzárendeli továbbá az elemekhez tartozó feliratokat, amelyek a „res\values\strings.xml” fájlban találhatók meg. Mivel minden felirat ebben a fájlban található meg, ha más nyelven szeretnénk megjeleníteni a feliratokat, elég csupán ezt a fájlt átírni. A program többi részét nem kell módosítani.

A program indításakor az alapértelmezett activity, azaz a DaNImain activity „onCreate()” metódusa hívódik meg, amelyet az Activity osztálytól örököl. A metódus betölti a „main.xml” fájlt, vagyis a kijelzőn megjelenik a szövegdoboz és a három gomb. A következő lépés a Bluetooth adapter aktiválása. A rendszer létrehoz egy eseményt az adapter aktiválására, amelyet az Androidban „intent”-nek nevezünk [18]. Különböző alkalmazások feliratkozhatnak különböző események kezelésére. A DaNImain által létrehozott eseménye hatására egy ablak jelenik meg, amely megkérdezi a felhasználót, hogy aktiválja-e az adaptert. Ha az adapter már aktív az alkalmazás indításakor, természetesen ez a lépés kimarad. A következő lépés a készülék gyorsulásmérőjének lefoglalása és beállítása. A program figyelőt rendel a szenzorhoz. Ez azt jelenti, hogy amint a gyorsulásiértékek a három tengely mentén változnak,

automatikusan meghívódik a figyelő „onSensorChanged(SensorEvent event)” metódusa. A paraméterként átadott „event” a SensorEvent osztály példánya. Tartalmaz egy „values” nevű tömböt, melynek első három eleme tartalmazza a gyorsulásértékeket a három tengely mentén. Ezekkel az adatokkal a metódus frissíti az „adat” nevű tömb értékeit. Ezt a tömböt küldi ki a program a robotnak. A továbbiakban, még mindig az onCreate() metóduson belül, létrehozuk a gombokat. Ezekhez is hozzárendelünk figyelőket a „setOnClickListener()” metódus segítségével. A „Csatlakozás” feliratú gomb létrehoz egy „BluetoothSocket” osztály példányt, amelyet beállít soros port profilra (SPP) a szolgáltatás UUID azonosítójának segítségével. Erre azért van szükség, mert a mobil készülékek Bluetooth modulja több szolgáltatást is tartalmaz. Ezekről a Bluetooth ismertetéséről szóló fejezetben lehet többet megtudni. A socket-ek segítségével ezekből a szolgáltatásokból kiválasztunk egyet. A Bluetooth kezelése a továbbiakban a létrehozott socket segítségével történik (csatlakozás, kapcsolat bontása és adatok küldése). Az SPP profil kiválasztására az Androidban UUID (Universal Unique Identifier) –t használunk. Ez egy 128 bites azonosító szám, amely a szolgáltatást azonosítja. A csatlakozáshoz szükséges a vevő modul Bluetooth MAC címe. Ezt a program már előre tartalmazza, így nem kell eszközöket keresni. A kapcsolódás előtt, az Android rendszerből a készüléket és a vevő modult párosítani kell. Ha mindez megtörtént, létrehozhatjuk a kapcsolatot az „mmSocket” objektum „connect()” tagfüggvényének meghívásával.

A „Kapcsolat bontása” gomb a már létrehozott mmSocket objektum „close()” tagfüggvényét meghívva bontja a kapcsolatot a vevővel.

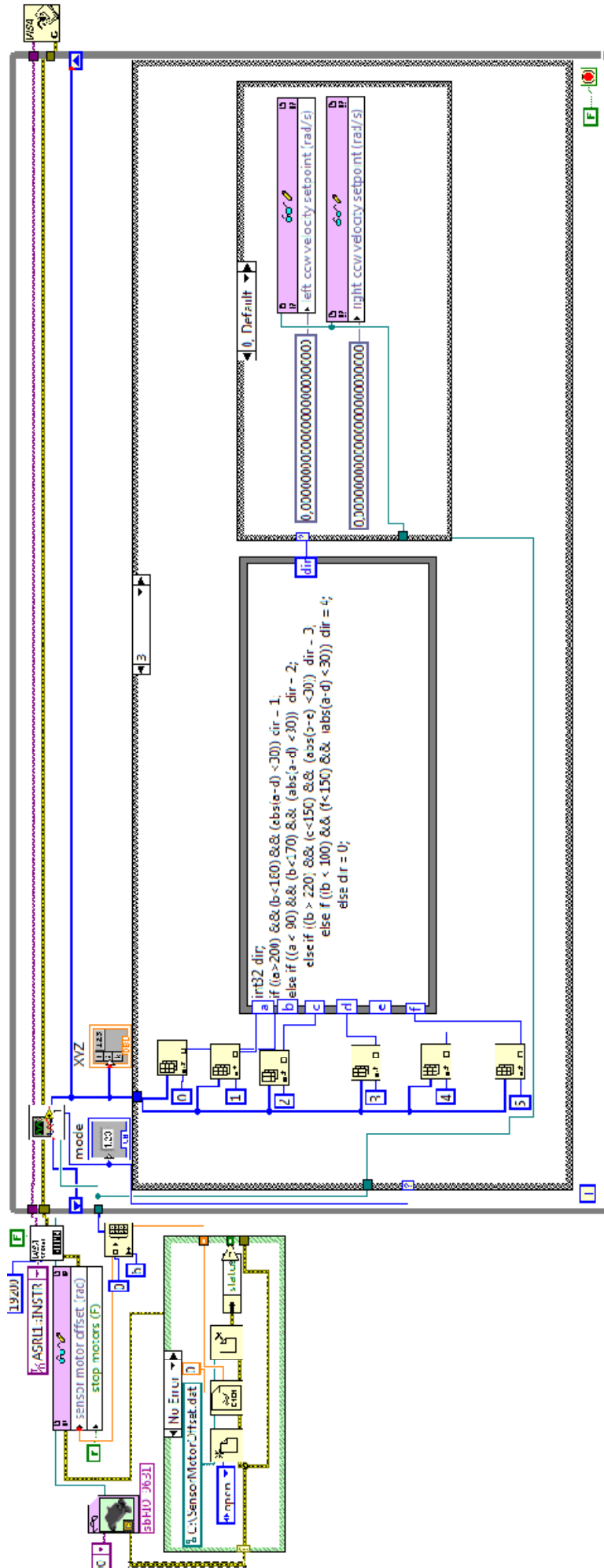
Az „Adatok küldése” gombot megnyomva indíthatjuk a kommunikációt a robottal. Ekkor létrehozunk egy OutputStream objektumot az „mmSocket.getOutputStream()” metódussal. Ezen objektum „send()” metódusával pedig küldjük az adatokat. A send() metódus futtatása a DaNImain osztály „run()” tagfüggvényében történik. Ez a tagfüggvény a Runnable interfészből származik, amelyet a DaNImain osztály implementál. A metódust úgy indíthatjuk, hogy a Thread osztályból példányosítunk („data_send” objektum) majd meghívjuk az objektum „start()” tagfüggvényét. A gombot ismét megnyomva leállítjuk az adatküldést. Az alkalmazást bezárva a rendszer automatikusan meghívja az „onDestroy()” metódust, amely bontja a Bluetooth kapcsolatot [19].

6. DaNI vezérlő programja

A robotban futó program a National Instruments által kiadott LabView Robotics program segítségével készült. A program egy „main.vi” és egy „get_data.vi” fájlból áll. Az előbbi a főprogram, az utóbbi pedig egy subVI, amely a kommunikációért felelős.

A 24. ábrán látható main.vi két fő részből épül fel: a program törzsét képező while ciklusból és a while cikluson kívüli inicializáló részből. Az inicializáló rész középre állítja a szervó motorra szerelt ultrahangos távolságmérőt. Az adatokat a számítógépen tárolt fájlból olvassa ki. A szervó kezdeti pozicionálását a robot összeszerelések és csatlakoztatásakor kell elvégezni. Ezt követi a robot soros portjának beállítása a LabView Visa moduljának segítségével. A programban ez a modul hajtja végre a soros portra vonatkozó összes műveletet.

A program törzsét képező while ciklus a kezdeti beállítások után a robot kikapcsolásáig fut. Első lépésként meghívódik a get_data subVI, amely a soros porton érkező adatok fogadásáért felelős. A gyorsulásmérőkből kiolvasott adatokon kívül ez a subVI választja ki a működési módot. Ha a robotot a mikrovezérlős távirányítóval irányítjuk, a subVI kimenete 1-től 3-ig változik a kiválasztott módnak megfelelően (a mikrovezérlős távirányítóval 3 módban irányíthatjuk a robotot). Symbian-os készülék esetén a kimenet a 4-es értéket, míg Androidos készülék esetén 5-ös értéket ad vissza. A mode értéke egy case struktúra választó feltétele, vagyis ez dönti el, hogy a kapott adatokat melyik programrészlet dolgozza fel. Az case struktúra öt esete a kapott adatokat megfelelő átalakítások után felhasználja a robot motorjainak vezérlésére. Ez a Read\Write Control függvény segítségével történik. A függvényen keresztül módosíthatunk illetve olvashatunk ki különböző adatokat az FPGA áramkörből. Példaként itt állíthatjuk be az egyenáramú motorok fordulatszámát, beállíthatjuk a szervó motor pozícióját, kiolvashatjuk az ultrahangos távolságmérővel mért adatokat, valamint hozzáférhetünk a panelon elhelyezkedő LED-hez.



24. ábra: main.vi

A 25. ábrán látható az adatok fogadását végző subVI. A program ezen része is két részből áll, amelyek Flat Sequence struktúrában helyezkednek el. A struktúra biztosítja hogy a két folyamat időben egymás után hajtódjon végre és ne párhuzamosan. Az első folyamatban kiürítjük a robot soros pufferét. Mivel a robotot irányító eszközök folyamatosan küldik a helyzetükre jellemző adatokat, ez biztosítja, hogy a kiszámolt mozgási sebesség és irány a legfrissebb beérkező adatokból lesz kiszámolva. Eután a program egy while ciklusba lép be, amelynek kilépési feltétele, hogy a fogadott bájt értéke 0, 1 vagy 2 legyen. Ezek jelzik az adatcsomag elejét és azonosítják a vezérlő eszközt.

A következő folyamat az adatok fogadása. 1-es vagy 2-es csomagkezdő bájt esetén (mobilkészülék, Symbian vagy Android operációs rendszerrel) a program 3 bájtot fogad, majd a mode változót 4-es vagy 5-ös értékre állítja be és visszatér a főprogramba. 0-ás csomagkezdő bájt esetén 6 bájtot fogad a program. Ez a mikrovezérlős távirányító csomagformátuma. Mivel ezen belül 3 különböző irányítási mód található, vizsgálni kell a kapott csomag első és negyedik bájtját. Ha ezek értéke egyszerre 255, módot kell váltani. Ekkor a robot panelján található LED annyiszor villan fel, ahányadik módban vagyunk. A kapott értékekkel és a mód sorszámaival a subVI-ból visszatérünk a főprogramba.

7. Fejlesztési lehetőségek

A vezérlők fejlesztésekor több irányban is el lehet indulni. Felmerült néhány probléma, amelyek megoldásra várnak. Az egyik ilyen probléma a robot és a vezérlő közötti kapcsolat bontásakor lép fel. Ha a vezérlőt állítjuk le előbb és a robotot nem, akkor a robot az utolsó kapott adatoknak megfelelően tovább mozog. Ennek oka, hogy bár a program kommunikációért felelős része „beragad”, ettől függetlenül fut az FPGA-ba programozott motorvezérlés. Ezen probléma megoldására át kell szervezni a robotban futó program működését. Egy másik fontos megoldandó feladat az akadályok észlelése. A roboton található ultrahangos távolságmérő segítségével észlelhető a robot előtti akadály. A programot úgy kell bővíteni, hogy ha a robot a vezérlőtől olyan parancsot kap, amely következtében az akadályba ütközne, azt ne hajtsa végre. További lehetőség még a vezérlő és a robot között megvalósítandó kétirányú kommunikáció. A jelenlegi megoldásban az adó eszközök folyamatosan küldik az adatcsomagokat, még akkor is, ha a mért értékek nem változtak. Nem biztosított hogy minden adatcsomag célba ér, sőt a robot soros puffer ürítésekor sok adat törlődik és nem kerül feldolgozásra. Ez nem is probléma ebben az esetben, mivel folyamatosan érkezik az adatcsomag. Egy másik megoldás lenne, hogy a vezérlő csak akkor küld adatot, ha valamelyik érték megváltozik. Ennek a megoldásnak az lenne az előnye az előbbivel szemben, hogy kevesebb adatot kell küldeni, ezáltal csökken a készülékek áramfelvétele. Ekkor viszont már szükség van kétirányú kommunikációra ellenőrzésként, hogy az adat valóban megérkezett-e. A mobilkészülékek és a Bluetooth vető esetében ez gond nélkül megoldható. Mikrovezérlős távirányítónál viszont a mikrovezérlő Rx bemenetét már felhasználtuk. Emiatt szoftveres úton kell megoldani a Bluetooth modul felől érkező adatok fogadását. A hardver módosítására nincs szükség, ugyanis gondolva a későbbi fejlesztésekre, a vezetékezés ki lett alakítva a panelon.

A dolgozat bemutatott egy alternatív módot a robotok irányítására. A kézmozgást felhasználva különböző módokban működtethető a robot. Egykezes irányítást valósítottak meg a mobilkészülékekkel kialakított vezérlők. Itt egy készülékkel állíthattuk be a mozgás irányát és sebességét. A készülék vízszintessel bezárt szögével arányosan, fokozatmentes sebességszabályozást valósít meg. A mikrovezérlős távirányító két áramköre kétkezes irányítást valósít meg. Itt két, egymástól független jelforrást felhasználva irányíthatunk, ezzel megnövelve az irányítási módok számát. Az itt megvalósított három módon kívül, az alkalmazási területnek megfelelően, más módok is kialakíthatók, amelyek jobban alkalmazkodnak a kívánt feladat végrehajtásához. A beprogramozott három mód közül az első kettő fokozatmentes irányítást valósít meg. Az első módban külön van választva a sebesség és az irány megadása, míg a második módban ezeket a két eszköz együttesen határozza meg. A harmadik mód pedig kifejezetten a két eszköz együttállására épül. Itt a sebesség állandó értékű, a mozgásirányt a két eszköz együttes pozíciója határozza meg. Kihasnálva a Bluetooth platformfüggetlenségét, különböző eszközökkel irányíthatjuk a robotot. Mivel a Bluetooth ma már nagyon elterjedt, bármilyen programozható készülékkel irányíthatunk. Emellett természetesen szükséges a gyorsulásmérő jelenléte, ha kézmozgással szeretnénk vezérelni a robotot. Láthattuk, hogy különböző operációs rendszerű mobilkészülékekkel és programozott áramkörrel is lehet irányítani.

Köszönetnyilvánítás



A szerző ezúton köszöni meg a Debreceni Egyetem HuComTech projekt részére az Európai Unió és a Magyar Állam támogatásával, az Európai Szociális Alap és az Európai Regionális Fejlesztési Alap társfinanszírozásában a TÁMOP 4.2.-08/1/0008-0009 számú, Az ember-gép kommunikáció technológiájának elméleti alapjai címet viselő pályázat támogatását.

A témában született tudományos cikkeim

1. Husi Géza (DE), **Tóth Attila (DE)**, Husi Géza (BME): **NI LABVIEW ROBOTICS STARTER KIT VEZÉRLÉSE KÉZMOZGÁS ALAPJÁN** 17th „Building Services, Mechanical and Building Industry Days” International Conference, 13-14 October 2011, Debrecen, Hungary
2. HUSI Géza (DE), SZÁSZ Csaba (UTC), **Tóth Attila (DE)**, TOBÁK Dénes (UTC), TÖRÖK Péter (UTC), ZSURZSA László (UTC), HUSI Géza (BME): **ABILITIES MULTI-MODAL COMMUNICATION DEVELOPMENT OF A PROTOTYPE ROBOT** INTERNATIONAL REVIEW OF APPLIED SCIENCES AND ENGINEERING Akadémiai kiadó Budapest. bírálólat alatt.

Irodalomjegyzék

[1] Dr Robert King – Mobile Robotics Experiments with DaNI

http://download.ni.com/pub/devzone/epd/mobile_robotics_experiments.pdf

letöltve: 2011.09.20.

[2] Labview bemutatása

<http://home.mit.bme.hu/~krebesz/oktatas/vimia312/labVIEW07meres3lab01.pdf>

letöltve: 2011.09.27.

[3] A Bluetooth rendszer

http://studio.pataky.hu/edu/14p/tavkozlesi_halozatok/Bluetooth.pdf

letöltve: 2010.10.04.

[4] Bluetooth 4.0

<http://halozatbazis.web4.hu/cikk/ket-bluetooth-40-chipet-jelentett-be-az-atheros>

letöltve: 2011.09.27.

[5] Bluetooth

http://www.mcl.hu/~fazek/mobil_infokom_oravazlat/oravazlat_bluetooth.ppt

letöltve: 2010.10.04.

[6] BTM-112 adatlap

http://www.sparkfun.com/datasheets/Wireless/Bluetooth/BTM112_wATcommands.pdf

letöltve: 2011.09.27.

[7] Avinash Gupta - Synchronous Serial Communication Tutorial – The Basics of I2C and SPI

<http://extremeelectronics.co.in/avr-tutorials/synchronous-serial-communication-tutorial--the-basics-of-i2c-and-spi/>

letöltve: 2010.06.23.

[8] Neveri Gábor – Egyszerű soros kommunikáció AVR-el (UART)

http://www.hobbielektronika.hu/kapcsolasok/egyszeru_soros_kommunikacio_avr-rel_uart_oldal2.html

letöltve: 2010.07.22.

[9] Wikipedia.org – RS-232

<http://hu.wikipedia.org/wiki/RS-232>

letöltve: 2011.09.28.

[10] Freescale Semiconductors - MMA7361 adatlap

http://www.freescale.com/files/sensors/doc/data_sheet/MMA7361L.pdf

letöltve: 2011.09.28.

[11] Topor Zoltán - Vezeték nélküli kommunikáció 434MHz-es (433,92 MHz) modulokkal

http://www.hobbielektronika.hu/kapcsolasok/434mhz_wireless.html

letöltve: 2011.09.28.

[12] Atmel – ATmega8 adatlap

http://www.atmel.com/dyn/resources/prod_documents/2486S.pdf

letöltve: 2010.06.20.

[13] Andreas Jakl - Symbian OS: Python/PyS60

<http://symbianresources.com/tutorials/general/python/Python.pdf>

letöltve: 2010.10.27.

[14] Charaf Hassan, Csúcs Gergely, Forstner Bertalan, Marossy Kálmán: Symbian alapú szoftverfejlesztés Szak kiadó, ISBN: 963-9131-66-0 2006

[15] A Python élete

http://www.hotdog.hu/magazin/magazin_article.hot?m_id=29143&a_id=602550&h_id=90719

letöltve: 2010.10.27.

[16] Forstner Bertalan, Ekler Péter, Kelényi Imre : Bevezetés a mobilprogramozásba

Szak kiadó, ISBN: 978-963-9863-01-09 2008

[17] Wikipedia.org – Android operációs rendszer

[http://hu.wikipedia.org/wiki/Android_\(operációs_rendszer\)](http://hu.wikipedia.org/wiki/Android_(operációs_rendszer))

letöltve: 2011.10.25.

[18] Android alkalmazások szerkezete

http://www.javaforum.hu/javaforum/8/android/android_cikkek/32/show/az_alkalmazasok_szerkezete

letöltve: 2011.10.25.

[19] Lauren Darcey, Shane Conder – Android Application Development in 24 Hours

Sams Publishing kiadó, ISBN: 978-0-321-67335-0

Mikrovezérlős áramkör programja

„A” áramkör (rf868_2.c)

```
#define F_CPU 16000000UL
#define BAUDRATE 9600UL
#define ELEM_MIN 100

#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

char bt_mac [] = {18, 65, 84, 68, 61, 48, 48, 49, 50, 54, 70, 48, 57, 70, 54, 68, 49,
13};
char connect[] = {5, 65, 84, 65, 13};
uint8_t XYZ[6];
uint8_t timer;
uint8_t bat;
uint8_t _0g;

void init_basic() {

    DDRD |= (1<<PD7) | (1<<PD6) | (1<<PD1) | (1<<PD5);
    DDRC |= (1<<PC5) | (1<<PC4);
    DDRB |= (1<<PB2);
    PORTC |= (1<<PC5);
    PORTD |= (1<<PD5);
}

void init_accelerometer() {

    PORTC |= (1<<PC4);
    //PORTD |= (1<<PD6); //érzékenység: 6g
}

void init_timer1() {

    TCCR1B |= (5<<CS00);
    TIMSK |= (1<<TOIE1);

}

void init_ext_interrupt() {

    MCUCR |= (3<<ISC00);
    GICR |= (1<<INT0);
}
```

```

void init_usart(unsigned int br) {

    DDRD |= (1<<PD1);
    UCSRB |= (1<<TXEN) | (1<<RXEN);
    UCSRC |= (1<<URSEL) | (1<<UCSZ1) | (1<<UCSZ0);
    unsigned int ubrr = F_CPU/16/br-1;
    UBRRH = (uint8_t) (ubrr>>8);
    UBRL = (uint8_t) ubrr;
}

void usart_send(uint8_t data) {

    while(!(UCSRA & (1<<UDRE)));
    UDR = data;
}

void usart_get_packet(uint8_t *data) {
    while(1) {
        while(!(UCSRA & (1<<RXC)));
        if (UDR != 'A') continue;
        while(!(UCSRA & (1<<RXC)));
        if (UDR != 'A') continue;
        for(uint8_t i=3;i<6;i++) {
            while(!(UCSRA & (1<<RXC)));
            data[i] = UDR;
        }
        while(!(UCSRA & (1<<RXC)));
        if (UDR != 'Z') continue;
        return;
    }
}

void usart_send_packet(uint8_t *data) {

    if (_0g) {
        data[0] = 255;
        _0g = 0;
    }
    usart_send(0);
    for(uint8_t i=0; i<6; i++) usart_send(data[i]);
}

void init_bluetooth() {

    PORTB |= (1<<PB2);
    _delay_ms(500);
    PORTB &= ~(1<<PB2);
    _delay_ms(4000);
}

```

```

void bluetooth_connect() {

    for(uint8_t i=1; i<bt_mac[0]; i++) usart_send(bt_mac[i]);
    _delay_ms(100);
    for(uint8_t i=1; i<connect[0]; i++) usart_send(connect[i]);
    _delay_ms(100);
}

void init_ADC() {

    ADMUX |= (1<<REFS0) | (1<<REFS1) | (1<<ADLAR);
    ADCSRA |= (1<<ADEN) | (1<<ADPS0) | (1<<ADPS1) | (1<<ADPS2);
}

uint8_t ADC_measure(uint8_t ch) {

    ADMUX = (ADMUX & 0b11110000) | (ch<<MUX0);
    ADCSRA |= (1<<ADSC);
    while(!(ADCSRA & (1<<ADIF)));
    ADCSRA |= (1<<ADIF);
    return(ADCH);
}

void check_bat() {

    ADMUX &= ~(1<<REFS1);
    ADC_measure(14);
    bat = ADC_measure(14);
    ADMUX |= (1<<REFS1);
    ADC_measure(14);
    if (bat>ELEM_MIN) {
        cli();
        PORTC &= ~(1<<PC4);
        while(1) {
            PORTC ^= (1<<PC5);
            _delay_ms(1000);
        }
    }
}

ISR(INT0_vect) {

    _0g = 1;
}

```

```

ISR(TIMER1_OVF_vect)
{
    if (timer<50) timer++;
        else {
            timer = 0;
            check_bat();
        }
}

int main() {
    init_basic();
    init_ADC();
    check_bat();
    init_accelerometer();
    init_bluetooth();
    init_ext_interrupt();
    init_usart(BAUDRATE);
    init_timer1();
    sei();
    bluetooth_connect();
    while(1) {
        TIMSK &= ~(1<<TOIE1);
        for(uint8_t i = 0; i<3; i++) {
            XYZ[i] = (ADC_measure(i));
        }
        usart_get_packet(XYZ);
        usart_send_packet(XYZ);
        TIMSK |= (1<<TOIE1);
    }
}

```

```
#define F_CPU 16000000UL
#define BAUDRATE 9600UL
#define ELEM_MIN 100

#include <avr\io.h>
#include <util\delay.h>
#include <avr\interrupt.h>

uint8_t XYZ[3];
uint8_t timer;
uint8_t bat;
uint8_t _0g;

void init_basic() {

    DDRD |= (1<<PD7) | (1<<PD6) | (1<<PD1);
    DDRC |= (1<<PC5) | (1<<PC4);
    PORTC |= (1<<PC5);
}

void init_accelerometer() {

    PORTC |= (1<<PC4);
    //PORTD |= (1<<PD6); //érzékenység: 6g
}

void init_timer1() {

    TCCR1B |= (5<<CS00);
    TIMSK |= (1<<TOIE1);

}

void init_ext_interrupt() {

    MCUCR |= (3<<ISC00);
    GICR |= (1<<INT0);
}

void init_uart(unsigned int br) {

    DDRD |= (1<<PD1);
    UCSRB |= (1<<TXEN);
    UCSRC |= (1<<URSEL) | (1<<UCSZ1) | (1<<UCSZ0);
    unsigned int ubrr = F_CPU/16/br-1;
    UBRRH = (uint8_t) (ubrr>>8);
    UBRRL = (uint8_t) ubrr;
}
```

```

void usart_send(uint8_t data) {

    while(!(UCSRA & (1<<UDRE)));
    UDR = data;
}

void send_packet(uint8_t *data) {
    usart_send('J');
    usart_send('A');
    usart_send('A');
    if (_0g) {
        usart_send(255);
        _0g = 0;
    }
    else {
        usart_send(data[0]);
    }
    usart_send(data[1]);
    usart_send(data[2]);

    usart_send('Z');
    _delay_ms(20);
}

void init_ADC() {

    ADMUX |= (1<<REFS0) | (1<<REFS1) | (1<<ADLAR);
    ADCSRA |= (1<<ADEN) | (1<<ADPS0) | (1<<ADPS1) | (1<<ADPS2);
}

uint8_t ADC_measure(uint8_t ch) {

    ADMUX = (ADMUX & 0b11110000) | (ch<<MUX0);
    ADCSRA |= (1<<ADSC);
    while(!(ADCSRA & (1<<ADIF)));
    ADCSRA |= (1<<ADIF);
    return(ADCH);
}

```

```

void check_bat() {

    ADMUX &= ~(1<<REFS1);
    ADC_measure(14);
    bat = ADC_measure(14);
    ADMUX |= (1<<REFS1);
    ADC_measure(14);
    if (bat>ELEM_MIN) {
        cli();
        PORTC &= ~(1<<PC4);
        while(1) {
            PORTC ^= (1<<PC5);
            _delay_ms(1000);
        }
    }
}

ISR(INT0_vect) {

    _0g = 1;
}

ISR(TIMER1_OVF_vect)
{
    if (timer<100) timer++;
    else {
        timer = 0;
        check_bat();
    }
}

int main() {
    init_basic();
    init_ADC();
    check_bat();
    init_accelerometer();
    init_ext_interrupt();
    init_uart(BAUDRATE);
    sei();
    init_timer1();
    while(1) {
        TIMSK &= ~(1<<TOIE1);
        for(uint8_t i = 0; i<3; i++) {
            XYZ[i] = ADC_measure(i);
        }
        TIMSK |= (1<<TOIE1);
        send_packet(XYZ);
    }
}

```

```
import appuifw, e32, axyz, socket, time
from socket import *

def read_xyz(x, y, z):
    dani.send(chr(1) + chr(x + 97) + chr(y + 97) + chr(z + 97))
    time.sleep(0.1)

def quit():
    axyz.disconnect()
    app_lock.signal()

address, services = bt_discover()
print services
print address
dani=socket(AF_BT,SOCK_STREAM)
dani.connect((address,services.values()[0]))
axyz.connect(read_xyz)

appuifw.app.title = u"DaNI control by Attila"
appuifw.app.exit_key_handler = quit
app_lock = e32.Ao_lock()
app_lock.wait()
```

```
package com.DaNI;

import java.io.IOException;
import java.io.OutputStream;
import java.util.UUID;

import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.Context;
import android.content.Intent;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class DaNImain extends Activity implements Runnable {

    @Override
    public void run() {
        while(flag == true) {
            try {
                outputStream.write(adat);
            } catch (IOException e) {};
            try {
                Thread.sleep(100);
            } catch (Exception e) {};
        }
    }

    private byte[] adat = new byte[4];
    boolean flag, first_time;
    private SensorManager sensorManager;
    private Sensor sensor;
    private static final int REQUEST_ENABLE_BT = 0x1;
    private static final UUID MY_UUID =
    UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");
    private BluetoothAdapter mBluetoothAdapter =
    BluetoothAdapter.getDefaultAdapter();
    private BluetoothSocket mmSocket;
    private final BluetoothDevice device =
    mBluetoothAdapter.getRemoteDevice("00:12:6F:09:F6:D1");
    private OutputStream outputStream;
    Thread data_send = new Thread(this);

    /** Called when the activity is first created. */
    @Override
```

```

        public void onCreate(Bundle savedInstanceState) {
            adat[0] = 2;
            flag = false;
            first_time = true;
            super.onCreate(savedInstanceState);
            setContentView(R.layout.main);
            if (!BluetoothAdapter.isEnabled()) {
                Intent enableBtIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
                startActivityForResult(enableBtIntent,
REQUEST_ENABLE_BT);}
            sensorManager = (SensorManager)
getSystemService(Context.SENSOR_SERVICE);
            sensor =
sensorManager.getSensorList(Sensor.TYPE_ACCELEROMETER).get(0);
            sensorManager.registerListener(accelerationListener,
sensor, SensorManager.SENSOR_DELAY_GAME);

            Button connect_but = (Button)
findViewById(R.id.button_connect);
            connect_but.setOnClickListener(new
View.OnClickListener() {

                @Override
                public void onClick(View v) {
                    // TODO Auto-generated method stub
                    try {
                        mmSocket =
device.createRfcommSocketToServiceRecord(MY_UUID);
                    } catch (IOException e) {};
                    try {
                        mmSocket.connect();
                    } catch (IOException e) {};
                }
            });

            Button disconnect_but = (Button)
findViewById(R.id.button_disconnect);
            disconnect_but.setOnClickListener(new
View.OnClickListener() {

                @Override
                public void onClick(View v) {
                    // TODO Auto-generated method stub
                    try {
                        mmSocket.close();
                    } catch (IOException e) {};
                }
            });

            Button send_data_but = (Button)
findViewById(R.id.send_data_button);

```

```

        send_data_but.setOnClickListener(new
View.OnClickListener() {

        @Override
        public void onClick(View v) {
            // TODO Auto-generated method stub
            if(first_time == true) {
                try{
                    outputStream =
mmSocket.getOutputStream();
                } catch (IOException e) {};
                first_time = false;
            }
            if (flag == false) {
                flag = true;
                data_send.start();
            }
            else {
                flag = false ;
                data_send.stop();
            }
        };

    });

}

private SensorEventListener accelerationListener = new
SensorEventListener() {
    @Override
    public void onAccuracyChanged(Sensor sensor, int acc)
{
    }

    @Override
    public void onSensorChanged(SensorEvent event) {
        adat[1] = (byte) (event.values[0]/2*10);
        adat[2] = (byte) (event.values[1]/2*10);
        adat[3] = (byte) (event.values[2]/2*10);
    }

};

@Override
protected void onDestroy() {
    // TODO Auto-generated method stub
    try {
        mmSocket.close();
    } catch (IOException e) {};
    super.onDestroy();
}

}

```