Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Többcsatornás adaptív streaming-et támogató távvezérlő rendszer megvalósítása Android alapú mobil kliensek kezelésével

Tudományos Diák konferencia dolgozat

2011

Készítette:

Jusztin Ádám adam.jusztin@gmail.com

Konzulensek:

Dr. Ekler Péter ekler.peter@aut.bme.hu

Dr. Csorba Kristóf kristof@aut.bme.hu

Tartalom

1. Bevezetés

Napjainkban a távvezérlésű eszközöket egyre szélesebb körben alkalmaznak. Ilyenek például a távszabályozások, biztonsági megfigyelések, lakások biztonságának és automatizálásának vezérlése, különböző járművek irányítása a földön és a levegőben egyaránt. A mobil telefonok rohamos fejlődésen mennek keresztül, melynek következtében képesekké váltak multimédiás tartalmak lejátszására továbbá az egyre több beépített funkciónak köszönhetően a legkülönfélébb feladatokra lettek alkalmasak. Ilyen alkalmazás lehet például a Google által nemrég bemutatott Android@Home [1] lakások automatizálásért felelős keretrendszer. Az okos telefonok és a tabletek elterjedésével egyre nagyobb az igény olyan távvezérlésekre, melyek ilyen készülékekről irányíthatóak.

Jelen TDK munka alkalmával olyan általános célokra is használható hardver és szoftver párost terveztem, mellyel vezeték nélkül irányítható egy berendezés valós idejű kép alapján. A tervezési elvek igazolására megvalósítottam egy prototípust, mely teljes mértékben megfelel a tervezett rendszernek. A tervezés és megvalósítás során mind a hardver oldali tervezés, mind pedig a szoftverfejlesztés területén számos egyedi komponenst kellett kidolgoznom, melyeket a dolgozatomban részletesen ismertetek. Ilyen megoldás például a WiFi router tápellátásának megoldása akkumulátorról.

A szakirodalomban fellelhető megoldásokhoz képest egy újszerű megvalósítás alapján a vezérlés központi egységéül egy WiFi routert alkalmaztam. A kliens oldali alkalmazást az egyre szélesebb körben alkalmazott Google által fejlesztett mobil operációs rendszerre, az Androidra implementáltam. Ez az alkalmazás képes a szerver által szolgáltatott valós kép megjelenítésére, továbbá az eszköz vezérlésére és a szükséges mérések elvégzésére és továbbítására. Ezen felül megvalósításra került az is, hogy egyszerre több kliens alkalmazás is csatlakozhasson és fogadhassa a távvezérlésű rendszer által szolgáltatott valós idejű képet, továbbá lehetőség van engedélyhez kötött vezérlés átadására is a felhasználók között, mely tovább növeli a felhasználási lehetőségeket. Ezekkel a megoldásokkal egy széles körben alkalmazható, rugalmas megoldást készítettem,mely több jelenlegi ipari és kereskedelmi eszköz vezérlésének továbbfejlesztési alapja lehet. Ilyen eszköz például a Parrot AR.Drone [2].

Minden valós idejű kép alapján történő vezérlésnél a kritikus pont a valós idejű kép késleltetése a felhasználó készüléknél. Ilyen helyzetekben sokkal fontosabb, hogy kicsi legyen a

valós kép megjelenítésének késleltetése minthogy kiváló legyen a megjelenített kép minősége. TDK munkámban többek között ennek a megoldását is célul tűztem ki. A kamera által látott és felhasználónál megjelenített kép késleltetésének minimálisra csökkentése végett, megterveztem és megvalósítottam egy adaptív videó streaming megoldást a beágyazott rendszerre, mellyel a felhasználói készüléknél mért tényleges WiFi kapcsolat jel-zaj viszonya és sebessége alapján történik az optimális videó stream ráta megválasztása. A megvalósított megoldást több szempont szerint is elemeztem az optimális képminőség elérése céljából.

Valós alkalmazásnál fontos ismerni, hogy a rendszer, hogyan reagál a különböző terhelési körülményekre, ezért a megvalósított rendszeren részletes teljesítményelemzést végeztem. Ez magában foglalja a szerver teljesítményének mérését: hány felhasználót tud kiszolgálni és milyen szolgáltatási minőség mellett. Ezen túlmenően kidolgoztam egy mérési metódust, mellyel az implementált adaptív algoritmus hatékonyságát vizsgáltam.

A következő fejezetben áttekintést adok a távvezérlő rendszer architektúrájáról. Ezután a 3. fejezetben a létező megoldások és a felhasznált technológiák alkalmazását mutatom be. A 4. fejezetben részletesen megmutatom az elektronika és tápellátás megvalósítását és kitérek a szerver és az elektronika közötti kommunikáció implementálására is. Az 5. fejezetben a rendszer szerver komponensét továbbá a szerver alkalmazást mutatom be. A 6. fejezetben az Android platformra megvalósított kliensalkalmazás részleteit ismertetem. A rendszer fejlesztése során nem akartam csak az Android platformra korlátozni a távvezérlést, ezért a 7. fejezetben bemutatom a platform független vezérlési megoldást is. Az adaptív videó streaming implementációt a 8.fejezetben. Végül a szerver és az adaptív algoritmus teljesítményét a 9. fejezetben vizsgáltam.

2. Áttekintés

A tervezett megoldás bemutatásához elsőként tekintsük át a rendszer magas szintű architektúráját. A teljes távvezérlő rendszer funkcionális blokk diagramja a 2.1. ábrán látható. 4 jól elkülöníthető részből tevődik össze. Az ábra bal oldalán az első egység a felhasználói készülékek, ezek lesznek a vezérlési céleszközök. A külvilággal a WiFi router teremt kapcsolatot. A routerhez kapcsolódik a webkamera és a vezérlő elektronika. A router szolgáltatja az élő képet a felhasználóknak és végzi a felhasználótól érkező vezérlési jelek fogadását és továbbítását a vezérlő elektronika felé. A vezérlő elektronika végzi a perifériák vezérlését, ami jelen esetben egy modell autó.

2.1. ábra: A távvezérlő rendszer funkcionális blokk diagramja

3. Létező megoldások és technológiák áttekintése

A munkám során tanulmányoztam a szakterület kapcsolódó irodalmat. A szakirodalom vizsgálata alapján arra jutottam, hogy sok megoldás foglalkozik jelen TDK egy-egy részletével azonban, ami újszerűvé teszi ezt a megoldást a többféle megoldás összekötése egy rugalmas modern rendszerré. A TDK kiindulásául a [3], [4] tanulmányokat választottam. Ezekben a tanulmányokban mobil eszközök irányítását mutatják be főleg kényes területeken alkalmazva mint például atomerőművek és katonai hadszínterek. Az irányítás interneten keresztül webes felületen történik. Ezen tanulmányokat megvizsgálva lehetőség adódott a továbbfejlesztésükre.
Ami az okostelefonok és tabletek elterjedésével elkerülhetetlenné vált, hogy a távvezérlő rendszereket lehessen ezekről a kézi készülékekről is irányítani, így ez volt az egyik irány a koncepciók továbbfejlesztésében. A vezérléshez felhasznált kliens alkalmazást a 6. fejezet részletezi. A másik felderített lehetőség pedig, olyan adaptív videó streaming használata, mellyel minimálisra csökkenthető az élőkép késleltetése, hiszen élő kép alapján kell vezérelni az eszközöket, így fontosabb az alacsony késleltetés mint a jó minőség. A [5] tanulmány is alacsony késleltetésű adaptív videó streaminggel foglalkozik. Ebben az implementációban a vezetékes internet kapcsolatra optimalizált adaptív algoritmust használnak. Azonban a vezeték nélküli kapcsolatok más algoritmusokat igényelnek. Így tanulmányoztam a vezeték nélküli kapcsolatokra optimalizált videó streaming megoldásokat is, és a következő két értekezést választottam kiindulási alapnak: [6], [7]. Az előbb említett publikációkat felhasználva egy vezeték nélküli kapcsolatokra optimalizált adaptív videó streaming megoldást valósítottam meg
A [8] tanulmány arra tér ki, hogy az okos telefonok számítási kapacitási folyamatosan növekszik ezzel együtt a készülékek teljesítmény felvétele is növekszik. Azonban ilyen mértékben nem növekedett az akkumulátorok teljesítménye, így igen fontos, hogy a mobil kliensre szánt alkalmazások optimalizálva legyenek, feleslegesen ne használják a telefon erőforrásait. A kliens alkalmazás fejlesztése során szem előtt tartottam ezt a szempontot is és próbáltam mindenhol törekedni a hatékony megoldásra.

A távvezérlő rendszerhez választott router, nem rendelkezett gyári firmware-rel, mivel ezek általában nem szabad hozzáférésű szoftverek, így a fejlesztők nem tudják kedvükre alakítani a rendszert. Hogy ezt a rugalmatlanságot kiküszöböljem egy nyílt forrású Linux-ot (OpenWrt [9]) alkalmaztam a routeren futó operációs rendszernek.

Végül az Android platformmal kellett megismerkednem és azokkal a komponensekkel, amikre a kliens alkalmazás implementálás során szükségem volt. Ezeket a megoldásokat részletesen bemutatom a dolgozatban

.

4. Elektronika megvalósítása

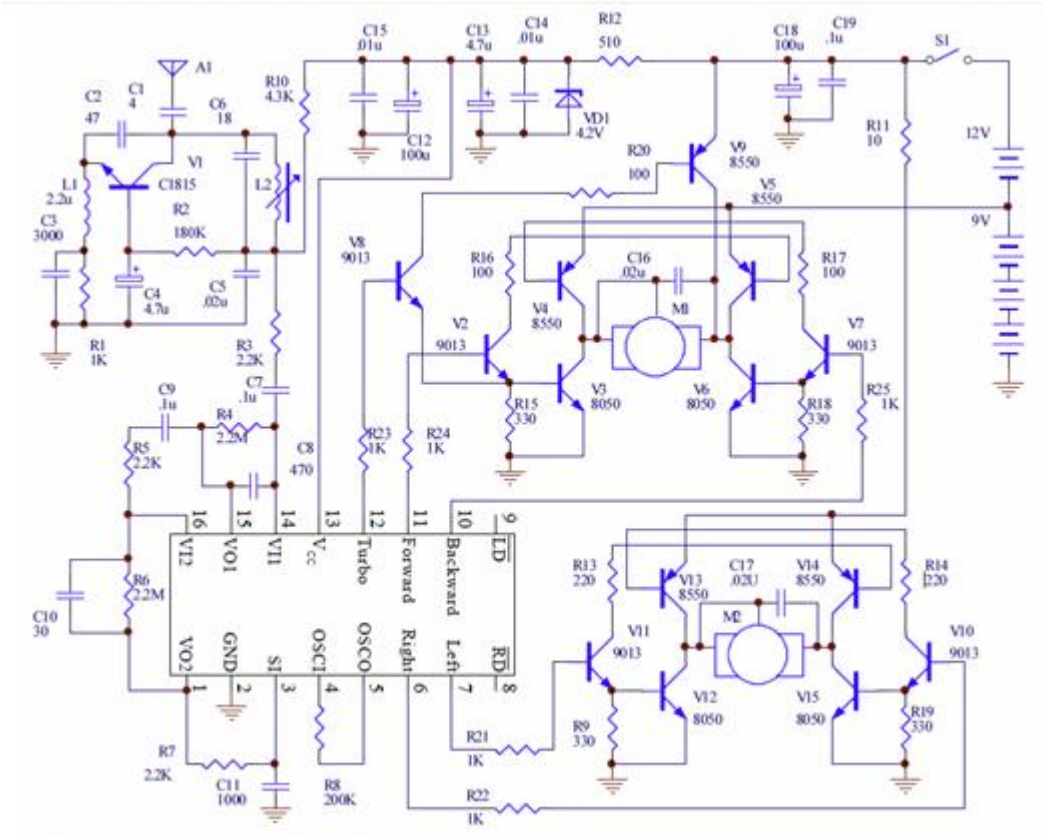
Az eddigiekben bemutattam a teljes rendszer architektúráját és a felhasznált technológiákat. A következő pontokban részletesen bemutatom az alkalmazott mikrokontrollert és a motorvezérlő elektronika illesztését is.

4.1. Mikrokontrollerek

A mikrokontroller egy IC-n megvalósított mikroszámítógép. Rendelkezik minden szükséges perifériával a működéshez, melyek a következőek processzormag, program memória (Flash), memória (RAM), I/O egységek (Input/Output), és az órajel képzéshez szükséges egységek. Ezeken felül rendelkezhetnek típustól függően különböző specializálódott egységgel is, mint például időzítők, kommunikációs vezérlők (UART, SPI, I^2C), A/D (Analog/Digital) illetve D/A konverterek. A mikrokontroller a Flash memóriába feltöltött utasítás sorozatot hajtja végre. Az utasítás sorozat legtöbbször egy főciklusból és megszakítás kezelő rutinokból épül fel. Megszakítással olyan feladat megoldása szükséges, mely nem várhat. Ilyen lehet tipikusan a kommunikációs interfészen érkező adat, amit azonnal el kell tárolni. A mikrokontrollerek számítási kapacitása jelentősen elmarad egy mai okos telefonétól is, azonban bizonyos feladatok ellátására így is remekül alkalmas. Ezek általában beágyazott rendszerek, ahol a fő szempontok az alacsony fogyasztás és a megbízhatóság.

4.2. Motormeghajtó elektronika

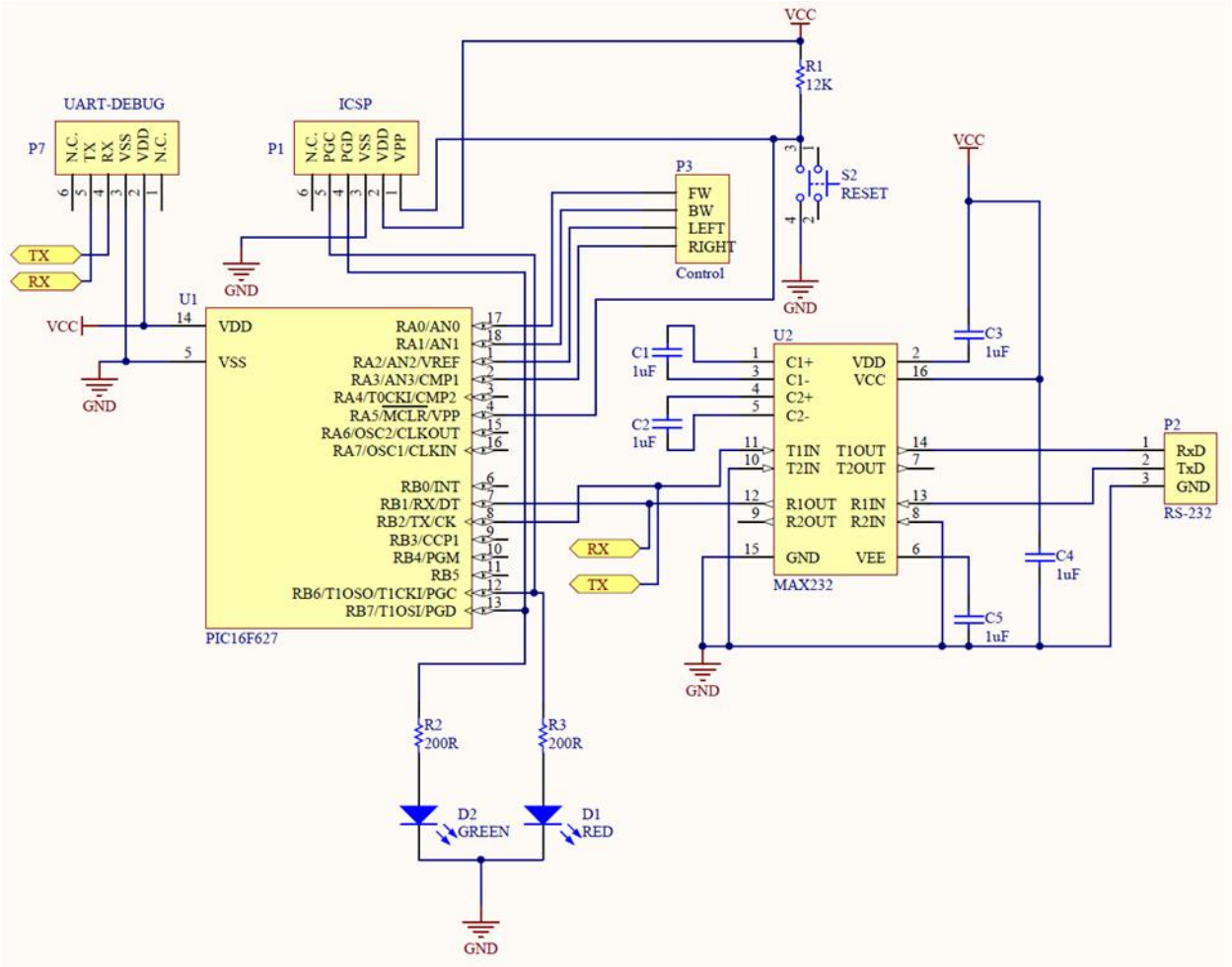
A vezérlő elektronika a routerhez kapcsolódik és ez az elektronika végzi a router által szolgáltatott vezérlő jelek feldolgozását. Szükséges a szerver oldali alkalmazás által feldolgozott eredmények továbbítása a cél hardver felé, mellyel a végső cél periféria vezérlése történik, ami jelen esetben a modell autó motorjai. Felhasználtam a modell autó motor vezérlő áramkörét az általam tervezett vezérléshez. A 4.1 ábrán látható gyári vezérlő elektronikából a DC motorvezérlő áramköri részeket használtam fel. A vezérlő IC jeleit ezentúl a vezérlő áramkör jelei helyettesítik. Ezek az ábrán a jobbra (right), balra (left), előre (forward) és a hátra (backward) jelek formájában láthatók.



4.1. ábra: a modell autó gyári elektronikája

4.3. Vezérlő elektronika

A router és a vezérlő elektronika közötti kommunikációt UART (Universal Asynchronous Receiver/Transmitter) valósítja meg. Ebből kifolyólag célszerű volt olyan mikrokontrollert választani, mely natív módon támogatja ezt a kommunikációt. Ennek megfelelően a választásom a PIC16F627-re [10] esett. A vezérlő áramkör kapcsolási rajza a **4.2**. ábrán látható. A központi egység (16F627) csupán a TTL szintű UART jeleket képes feldolgozni, ezért szükség volt az RS-232 +/- 12V-os szintjét egy szintillesztővel a mikrokontroller által megkívánt TTL szintre hozni. Erre a feladatra a Maxim MAX232 IC-jét választottam [11]. Továbbá szerepel még a kapcsolási rajzon egy ICSP (In Circuit Serial Programming) csatlakozó, mely a PIC programozásáért és a hibakeresési lehetőség biztosításáért felelős. Az áramkör huzalozási terve a melléklet 12.3. ábráján látható. A tesztelés során felmerült az UART szintű hibakeresés szükségessége is, ezért kiegészítettem az áramkört ezzel a lehetőséggel.



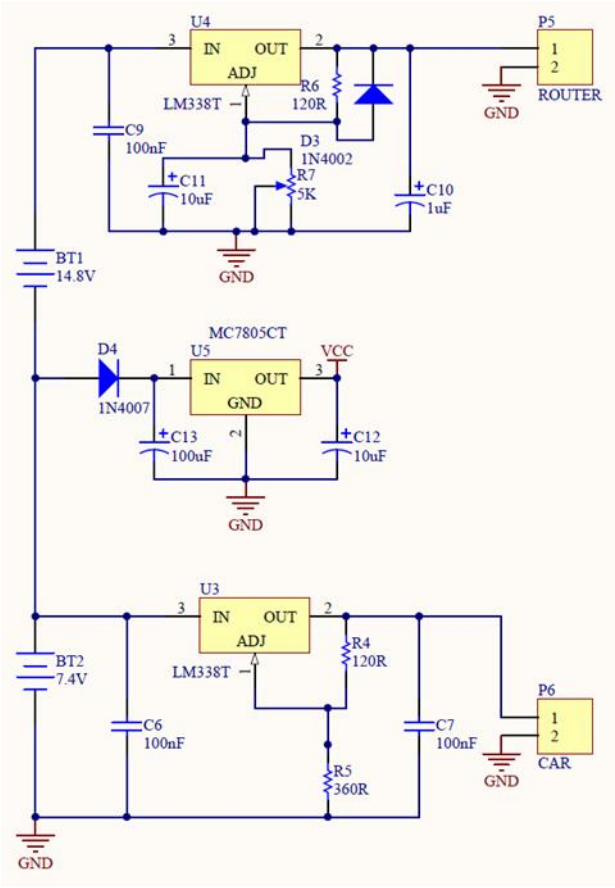
4.2. ábra: a vezérlő elektronika kapcsolási rajza

A 4.1. ábrán látható gyári áramkör irányító jelek vezérlését veszi át a 4.2. ábrán látható elektronika a *Control* csatlakozón keresztül (*FW, BW, LEFT, RIGHT*). Az eredeti vezérlő IC nem volt képes a motor fordulatszámát vezérelni, azonban az általam tervezett áramkörrel ez is lehetségessé vált.

4.4. Tápellátás

A tervezés során a tápellátás megoldása érdekes problémakör volt, hiszen a standard megoldások mellett meg kellett oldanom a router akkumulátorról való üzemeltetését is, mely önmagában is számos kihívást magában rejtő feladat. A teljes rendszer tápellátás szempontjából 3 részre bontható. A router számára 18V-ot, a mikrokontrollernek 5V-ot valamint a modell autó motorjainak is 5V-ot kell biztosítani. Ezen feszültségeket a 4.3. ábrán látható IC-kel valósítottam meg, felhasználva a gyártók által biztosított referencia áramköri terveket. A nagyobb

teljesítmény igényű részekre a modell autó és a router tápellátásának biztosítására az LM338-as szabályozható feszültség stabilizáló IC-t használtam, mely a névleges 5A-es kimeneti árammával ideális a router és a motorok számára. Az vezérlő elektronika számára elégséges volt az 1A-t biztosító 7805-ös feszültség stabilizáló IC használata. Tápforrásnak 6 db egyenként 3.7V-os és 4000 mAh-ás LiPo (Lithium-Ion Polymer) akkumulátorokat alkalmaztam. Mivel a rendszer hordozójául választott modell autó nem bírta megmozdítani a rendszert, hasonló teljesítményű, de jóval nehezebb akkumulátor alkalmazása esetén.

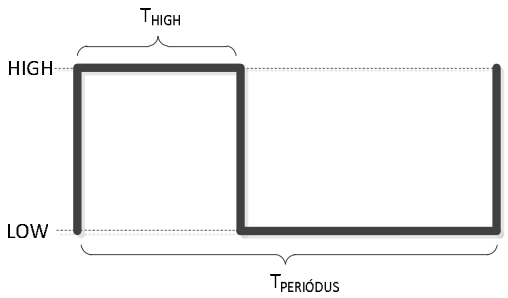


4.3. ábra: tápellátást biztosító áramkör

4.5. Fordulatszám szabályozás

A motor fordulatszám szabályozását PWM-mel (Pulse-Width Modulation) valósítottam meg. A frekvencia kiválasztása befolyásolja a tranzisztorok kapcsolási sebességét és a motor által kiadott hangot. Ennek megfelelően ezt 4kHz-re választottam, hogy minimális legyen az emberi hallás számára érzékelhető, ezáltal zavaró zaj. Adott frekvencián a kitöltési tényező

változtatásával változtatható a motorra jutó feszültség effektív értéke, ezáltal változtatva a motor fordulatszámát.



4.4. ábra: PWM jelalak

A 4.4. ábrán látható jelölésekkel a $T_{periódus}$ az 1. egyenlettel számolható, ahol az f a jel frekvenciája. Ekkor a kitöltési tényező (*DC* - Duty Cycle) a 2.egyenlettel határozható meg.

$$T_{periódus} = \frac{1}{f} \tag{1}$$

$$DC = \frac{T_{HIGH}}{T_{periódus}} \tag{2}$$

A fordulatszám szabályozás felbontását 4 bitben határoztam meg. Ez 16 fordulatszám értéket jelent. A jelen alkalmazás szempontjából ilyen finomságú felbontás tökéletesen elégséges.

4.6. Kommunikáció

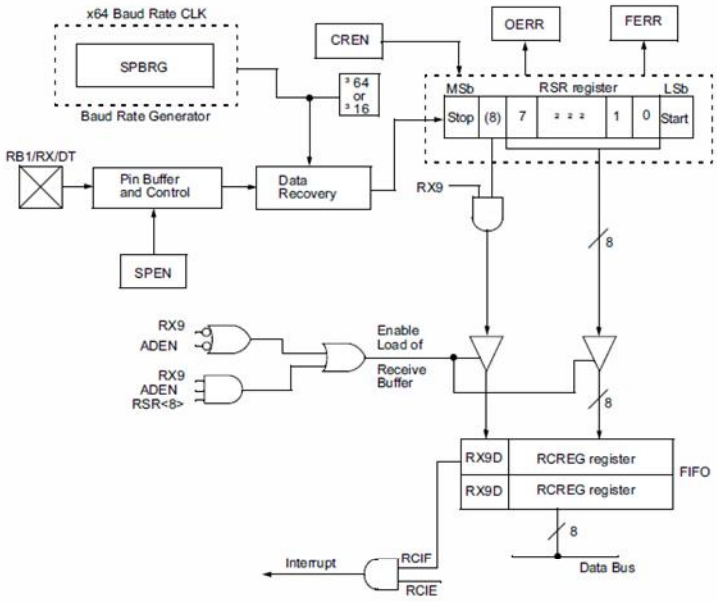
A router és a vezérlő elektronika között UART-al (Universal Asynchronous Receiver/Transmiter - univerzális aszinkron adó-vevő) történik a kommunikáció. Az UART egy soros interfész, mely számítógép és külső perifériák vagy mikrokontrollerek közötti kommunikációra alkalmas. A nevéből is adódik, hogy aszinkron módon történik az adatok küldése fogadása, tehát a kommunikációs felek között előre egyeztetni kell, hogy milyen mintavételezési időt alkalmazzanak. A mintavételezési sebességet baud rátának nevezik, a szabványos értékei 9600, 18200, 38400, 57600 és 115200 bps. Így ellehet érni, hogy a föld

mellett csupán 2 vezeték - egy-egy irányonként - keljen a kommunikációhoz. A jelvezetékeket általában RX/TX-nek nevezik (receive és transmit).

A kliens alkalmazás TCP-n keresztül küldi a szerver alkalmazásnak a beépített gyorsulásmérő szenzor aktuális értékeit, ezekhez az értékekhez egyenletes skálán hozzárendeltem egy fordulatszám értéket a 0 és a 100% fordulatszám között a fent meghatározott 4 bit finomságban. Így 16 sebességfokozat lett elérhető a vezérléssel. Sokkal nagyobb finomságot is lehetne alkalmazni, azonban ez nem vált szükségesség az alkalmazás szempontjából (8 bit esetén már 256 sebességfokozat).

4.7. PIC UART mód

Először be kell állítani az UART kommunikációhoz szükséges paramétereket. Ebben a módban a PIC a standard NRZ (non-return zero) módot használja: egy START nyolc adat és egy STOP bitből áll egy szó. Az aszinkron kommunikáció a baud rátáját 9600bps-ra választottam, mivel elégséges ez az adatátviteli sebesség a 2 bájtos vezérlőszavak átvitelére. A PIC16F627 hardveresen nem támogatja a paritás bitek használatát, így ezt a kommunikáció során nem alkalmazom. Az aszinkron kommunikációt az alábbi fő részek közreműködésével jön létre: baud rate generátor, mintavételezési áramkör, aszinkron adó/vevő. Részletesen az aszinkron vevő blokk vázlatát mutatom be a 4.5.ábra



4.5. ábra: UART vevő blokk diagram

A vevő egyik legfontosabb része a vevő shift regiszter (*RSR*- Receive Shift Register). A STOP bit mintavételezése után az *RSR* tartalma átíródik az *RCREG* regiszterbe. Ha az átvitel teljes akkor az *RCIF* bit jelzi ezt. A legfontosabb lépések, az UART mód beállításához:

1. SPBRG beállítása a megfelelő baud rátára

   movlw 0x19 ;0x19=9600 bps

   movwf SPBRG

2. aszinkron soros port engedélyezése: a *SYNC* bit nullázása és az *SPEN* bit beállítása

   movlw b'00100100'

   movwf TXSTA

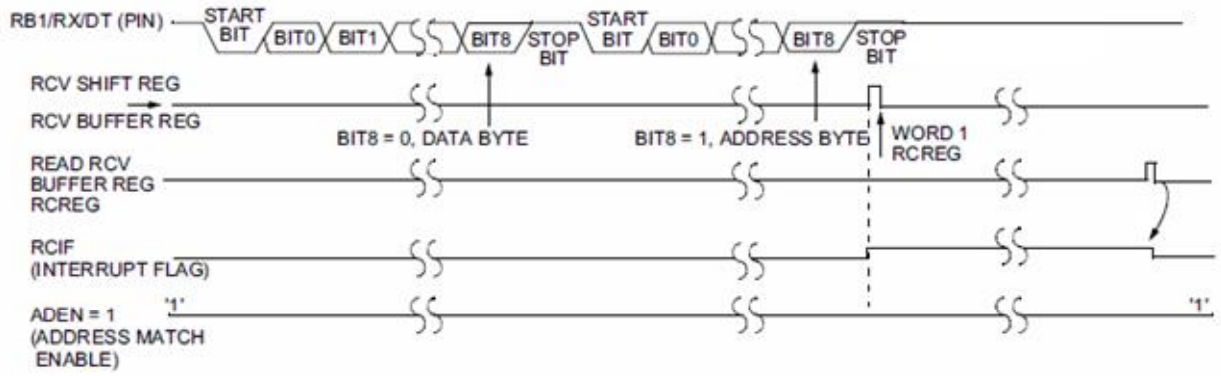
3. fogadás engedélyezése *CREN* bit beállításával

   movlw b'10010000'

   movwf RCSTA

4. *RCIF* bit jelzi, hogyha a szó fogadása sikeres

A 4.6. ábra a mintavételezést mutatja be aszinkron vétel esetén.



4.6. ábra: aszinkron vétel mintavételezése

4.8. PIC vezérlési algoritmus

A 4.7. fejezetben ismertetett UART mód helyes beállítása után szükség van még a kimenetek és bementek definiálására. A 16F627-en két portba vannak csoportosítva a lehetséges ki/bemenetek. Az alábbi kódrészlettel történik a megvalósítás:

1. "A" port beállítása

   movlw b'00000000'

   movwf PORTA

2. RB2(TX) = 1 az UART kommunikáció küldési kimenetének beállítása, a többi 0
   movlw b'00000100'
   movwf PORTB
3. "A" port összes lába kimenet
   movlw 0x00
   movwf TRISA
4. a "B" port lábai kimenetek kivéve RB1(RX)=bemenet, ez lesz az UART kommunikáció fogadási bemenete
   movlw b'00000010'
   movwf TRISB

A fenti beállítások után, a PIC várja a router soros porton át küldött "kész vagyok (READY)" üzenetét. Erre azért van szükség, mivel a router bootolás közben különböző hibakeresési célokkal folyamatosan írja a soros portra az éppen inicializált hardver állapotát, így READY üzenet nélkül nem lehetne pontosan megmondani, hogy mikortól van kész a router a bootolással és mikortól tud valós, a felhasználótól származó vezérlési információkat küldeni. Innentől kezdve a router és a vezérlő áramkör is készen áll a felhasználók kiszolgálására és a vezérlési parancsok fogadására. A vezérlő szó 2 bájtból áll. Az első megmondja az irányt a második pedig a sebességet illetve az irányváltás mértékét. A választott modellautó irányváltása nem szervóval hanem egyszerű DC motorral "ütközésig" történik, azonban a vezérlő elektronika képes kezelni a szervó motor irányváltásához szükséges jeleket is.

4.9. Tesztelés

Az elektronikát teszteltem még nem beépített állapotban is (12.2. ábra). A legfőbb szempontok voltak az UART kommunikáció és a motorvezérlő jelek vizsgálata. Az UART kommunikáció bit szintű tesztelése során kiderült, hogy a szerver oldali program, nem megfelelő EOL (end of line - sorvége karakter) karakterrel jelzi egy-egy bájt elküldését, így a mikrokontroller nem érzékelte, hogy a szerver befejezte az adatok küldését. Ezt Linux oldalon a soros port megfelelő beállításával (*stty*) lehetett orvosolni.

Miután ráépítettem a vezérlő elektronikát a hordozó platformra továbbá a routert is az autóra erősítettem a teljes eszköz a 12.1. ábrán látható. Az összeépítés után teszteltem a WiFi adó hatótávolságát, az elérhető sebesség tartományt és az áramfelvételt. Az elérhető

sebességtartománynak a 4 bites PWM felbontás szab határt. Tehát így 16 sebesség fokozatot lehet elérni, a 0% és 100% között egyenletesen elosztva. A teljes rendszerrel maximálisan 1.4 m/s érhető el. A tesztelés során megmértem a teljes rendszer áramfelvételét komponensekre lebontva ez látható 4.1. táblázatban. A mérések során 800x600 pixel méretű valós idejű videóképet használtam, így körülbelül 30 Mbps videó stream rátát mértem (a ráta változhatott a tartalom alapján).

Sebesség [m/s]	Adatforgalom [Mbps]	Áramfelvétel [A]	
		Router 18V	Motorok 5V
0 nyugalmi	~30	0.63	0
0 kerekek blokkolva	~30	0.63	2.35
1.4 maximális	~30	0.64	1.75

4.1. táblázat: a rendszer áramfelvétele

A WiFi hatótávolságának mérését szabadban végeztem. Így mobil telefont használva kliensnek (HTC WildFireS) 240 métert mértem. Ennél távolabb már nem volt valós idejű kép. Ennél a távolságnál is a mért átviteli sebesség 1 Mbps alá esett. Ennél a mérési elrendezésnél a telefon WiFi teljesítménye és az antennája volt a korlátozó tényező. Laptoppal és a laptophoz csatlakoztatott külső antennával (TP-LINK 2.4Ghz 5dBi), az elért távolság megduplázódott körülbelül 540 métert mértem 1 Mbps mellett.

5. Szerver

A megoldás komponenseinek ismertetésében a következő lépés a szerver bemutatása. A felhasználó és a vezérlő elektronika közötti kapcsolatot és a kommunikációt a korábban említett WiFi router végzi, melyen egy 680 Mhz-es Atheros CPU található és 128 MB RAM (DDR). Továbbá 3 mini-PCI csatlakozó, melyekből egyet használok a WiFi kártyának (54 Mbps-os névleges sebesség). A routeren futó operációs rendszernek az OpenWrt alkalmaztam [9]. Az OpenWrt egy Linux disztribúció kifejezetten beágyazott rendszerekre tervezve, így nagyfokú irányíthatóságot biztosít a router fölött. Az OpenWrt-nek igen alacsonyak a rendszer követelményei, és ezeket a követelmények az általam használ router teljesíti. A szerver teljesítménye az alkalmazásom szempontjából tökéletesen megfelelő. A szerver részletes teljesítmény elemzését a 9. fejezetben mutatom be részletesen.

5.1. Vezérlés

Ahhoz hogy felhasználók csatlakozhassanak a szerverhez és vezérelhessék az eszközt szükség van egy TCP szerverre mely a 6.3. fejezetben megfogalmazott vezérlés átadás miatt képes kell legyen arra, hogy csupán egy meghatározott IP címről fogadja el a vezérlési parancsokat. Erre a célra a netcat-et választottam [12]. Ez egy egyszerű alkalmazás mely tud olvasni és írni hálózati kapcsolatok között TCP vagy UDP protokoll használatával. Miután már az alkalmazás tudja fogadni a felhasználók által küldött parancsokat, már csak arról kell gondoskodni, hogy a szerver által fogadott vezérlési parancsok továbbítva legyenek a soros portra. Ebből a célból egy C programot írtam, melynek a feladata hogy a vezérlési jeleket elküldje a soros portra, a megfelelő formátumban [13]. A program működéséhez fontos részeket röviden ismertem. Először szüksége van egy handle-re ami kezeli majd a soros portot:

```
serial_port=CreateFile(/dev/ttyS0,GENERIC_READ|GENERIC_WRITE,0,0,OPEN_EXISTING,0, 0);
```

Ezután a vezérlő elektronika által megkívánt (4.7. fejezet) soros port paraméterek állítandók be.

```
void set_up_serial_port(HANDLE h)
{
  properties.BaudRate = CBR_9600; //9600 bps
  properties.Parity = NOPARITY;  //nincs paritás bit
  properties.ByteSize = 8;             //8 adat bit
  properties.StopBits = ONESTOPBIT; //1 STOP bit
  SetCommState(h, &properties);
  return;
```

```
}
```

Ezek után kerülhet sor a fájl soros portra írásának. Jelen esetben a fájl csupán 2 bájtból áll. Az első jelöli az elektronikának az irányt, a második pedig az irányváltás mértékét.

```
void write_file_to_serial_port(HANDLE h, char *file_name, unsigned long file_size)
{
  FILE *control_file;
  unsigned long bytes_rem = file_size;
  unsigned long bytes_sent;
  unsigned long bytes_read;
  unsigned long total_bytes_sent = 0;
  unsigned long bytes_to_send = 2;
  char buffer[10];

  /* fájl megynitása */
  control_file = fopen(file_name, "rb");

  /* ha nem tudja megnyitni kilép*/
  if (control_file == NULL)
  {
    fprintf(stderr, "ERROR%s\n", file_name);
    exit(0);
  }
  while (true)
  {
    /* meghatározott számú karakter olvasása*/
    bytes_read=(unsignedlong)fread((void *)buffer,1,bytes_to_send, control_file);

    /* adatok küldése*/
    WriteFile(h, (void *)buffer, bytes_read, &bytes_sent, NULL);
    if (bytes_sent != bytes_read)
    {
      CloseHandle(h);
      exit(0);
    }
    /* -- hátralévő bájtok száma*/
    bytes_rem -= bytes_sent;
    /* ++ elküldött bájtok*/
    total_bytes_sent += bytes_sent;
  }
  fclose(control_file);
  return;
}
```

5.2. Cross compiling

A cross compiler egy olyan eszközkészlet, mellyel futtatható állományokat lehet készíteni egy másik platformra mint amin az eszköz készlet fut. Tipikusan a beágyazott rendszerekhez használható. De amikor a kliens alkalmazást implementáltam Android platformra,

akkor is a PC-re telepített cross compliler végezte el az ARM proceszorra való optimalizálást. Fejlesztés során erre a célra a BuildRoot-ot alkalmaztam [14], hogy a C kódokból a routeren futtatható állományok legyenek.

5.3. Videó streaming

A valós képet egy webkamera szolgáltatja. Az általam választott kamera eszköz maximális felbontása 800 x 600 pixel. A kamera USB2-es csatlakozón kapcsolódik a routerhez. A szerveren szükséges ezek után a webkamera által szolgáltatott kép feldolgozása és továbbítása egy meghatározott adatfolyamban. Erre a célra a webkamerák által szolgáltatott natív adatfolyamot alkalmaztam mely az MJPEG (Motion JPEG). Az MJPEG jellegzetessége, hogy egymás után továbbított és egyedileg tömörített JPEG képeket tartalmaz, megfelelő HTTP fejléccel ellátva [15]. Az MJPEG stream továbbítására az mjpeg-streamer alkalmazást használtam [16], melyet megfelelően ki kellett egészítenem, hogy képes legyen valós időben változtatni a JPEG képek tömörítését és így a videó stream bit rátáját változtatni, továbbá, hogy HTTP get parancsokkal lehessen vezérelni a tömörítési arányt ezzel megvalósítva az adaptív videó stream lehetőségét.

5.4. Tömörítés

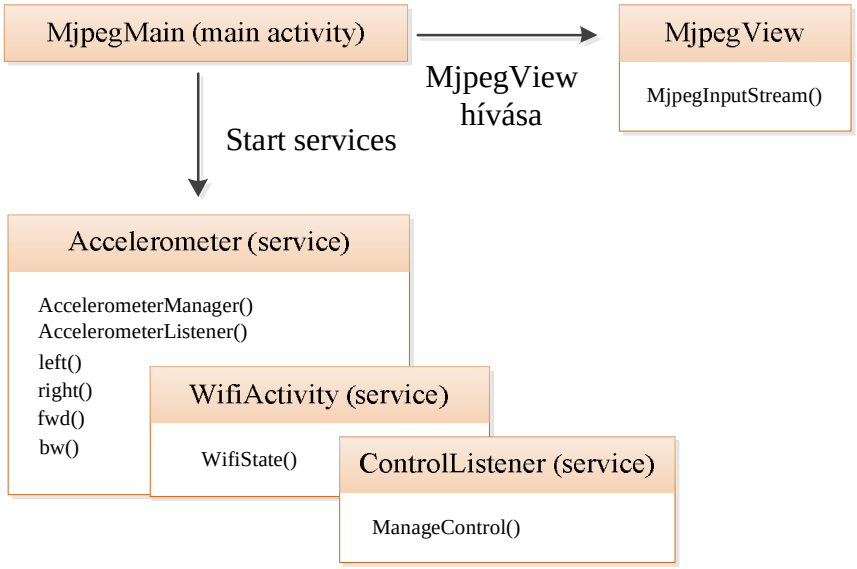
Az MJPEG-nél veszteséges intraframe[1] tömörítés alkalmazható mely a DCT-n (Discrete Cosine Transform) alapszik [17]. A DCT a transzformációs kódolás csoportjába tartozik, melynek célja a bemeneti jel egy hatékonyabb koordináta rendszerben történő felírása. Így az új koordináta rendszerben a jelminták közötti korreláció kisebb, mint az eredetiben, melyből következik, hogy a transzformációnak redundancia csökkentő szerepe van. A veszteséges tömörítés kihasználja az emberi látórendszer tulajdonságait, vagyis az emberi szem a kép nagyfrekvenciás részleteit kevésbé veszi észre, tehát ebben a tartományban keletkezett kvantálási zajra kevésbé érzékeny. A DCT kódolás egyik mellékhatása az un. blokk-effektus, amely alacsony bitsebesség mellett jelentkezik és a blokkok egymástól való független feldolgozásának eredménye. JPEG tömörítésnél a kódolásért felelős egységet [18] a Q paraméterrel lehet vezérelni. Ez a Q paraméter a JPEG kép tömörítés utáni minőségét jelenti, ahol az 1-es az

[1] képen belül önmagában kódolt, pl MPEG-4-nél van képek közötti prediktív kódolás

elérhető legrosszabb minőségű képet eredményez, a 100-as érték pedig a legjobb minőségű JPEG képet eredményezi (persze ez sem lesz veszteségmentes) [17].

6. Android alapú kliens alkalmazás

Ebben a fejezetben bemutatom az Android alapú kliens alkalmazás megvalósítását. Az implementált osztályokat röviden ismertetem és részletesen ismertetem a kulcsfontosságú komponenseket.

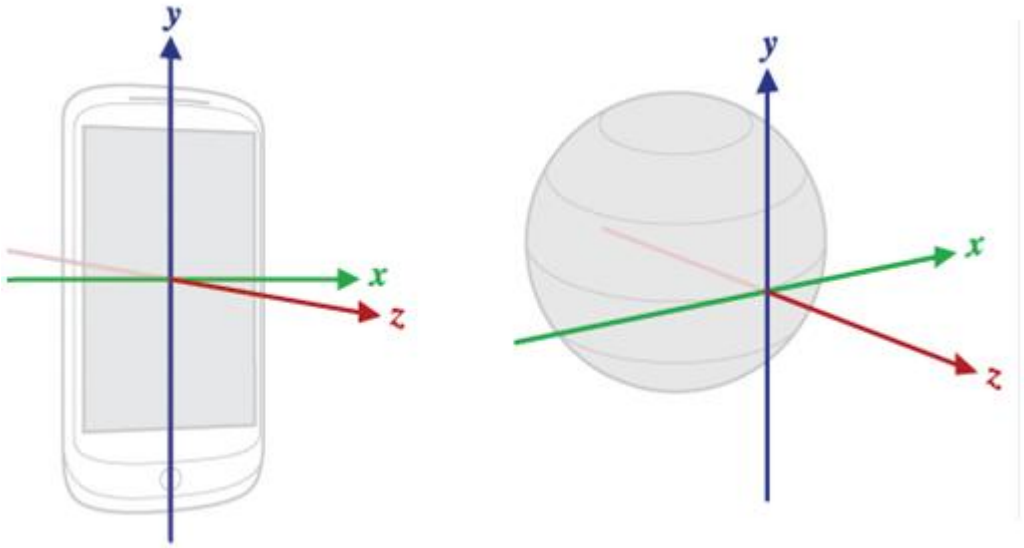


6.1. ábra: a kliens alkalmazás felépítése

Osztály	Felelősség
MjpegMain	Kezdő activity, mely meghívja a service-eket és az *MjpegView*-t
MjpegView	Az mjpeg stream megjelenítését végzi
MjpegInputStream	A szerverhez való kapcsolódást, a videó stream fogadását és a HTTP header-ek felbontását végzi
Accelerometer	Egy service, mely felelős az irányítással kapcsolatos adatgyűjtési és vezérlési feladatokért
AccelerometerManager	Az orientációs szenzor olvasása
AccelerometerListener	Feliratkozás az *AccelerometerManager* által szolgáltatott orientációs információkra

Left *Right* *Fwd* *Bw*	A megfelelő irányhoz tartozó vezérlő szó kiküldése a szervernek
WifiActivity	A *ConnectivityManager*-től a WiFi RSSI[2] érték és a kapcsolat sebességének lekérdezése, amikor változás van az értékben
WifiState	Az RSSI értékek továbbítása a szervernek az adaptív streaming miatt
ControlListener	A szerverre való bejelentkezést végzi és figyeli, hogy érkezik-e valamilyen *CONTROL REQUSET* jelek
ManageControl	Az érkezett *CONTROL REQUEST* jelek feldolgozása és a megfelelő esemény elindítása

6.1. Gyorsulás mérő és az irányítás



6.2. ábra: az orientációs szenzor használatának értelmezése

A legtöbb okos telefon rendelkezik beépített gyorsulásmérő szenzorral. Ezt a szenzort használja az Android például a képernyő orientáció megváltozásának detektálására. A gyorsulásmérőt sok játék és egyéb alkalmazás (widgedt-ek) használják intuitív irányításra. Ezért én is alkalmaztam a távvezérlésű autó irányítására. A telefon mint egy botkormány funkcionál és

[2] Received Signal Strength Indicator: Fogadott jel erősségének mutatója

a megfelelő irányokba döntés esetén, ennek paramétereit a szervernek elküldve, az autó a megfelelő irányba mozdul el.

Az Android operációs rendszer lehetőséget biztosít az érzékelők értékének viszonylag egyszerű lekérdezésére. A szenzorok lekérdezésére (amikor történik változás az értékükben) jól alkalmazható megoldás lehet egy háttérben futó service alkalmazása. A szenzor olvasása egyszerű megoldható az alábbi kódrészlettel:

```
public static void startListening(AccelerometerListener accelerometerListener){
sensorManager =
(SensorManager) Accelerometer.getContext().getSystemService(Context.SENSOR_SERVICE);
List<Sensor> sensors = sensorManager.getSensorList(Sensor.TYPE_ORIENTATION);}
```

Az X értéke az északi pólus iránya és az y tengely által bezárt szög a z tengely körül, ennek értéke 0°-tól 359°-ig terjed. Az Y értéke az x tengely körüli forgatás értéke (-180°-tól 180°-ig). A Z értéke pedig az y tengely körüli forgatás értéke (-90°-tól 90°-ig) [19].

Az alkalmazásomban az *Accelerometer* osztály végzi a *SensorEventChange* események észlelését és az *Accelerometer* osztályban rendeltem az (X,Y,Z) érték hármasokhoz egy megfelelő irányt. Amennyiben a kliens alkalmazásnál engedélyezve van az irányítás, akkor ezekhez a hozzárendelt irányokhoz tartozó vezérlő szavak kiküldésre kerülnek. Ezt 4 byte-os formában határoztam meg, ahol az első az irányt jelenti a másik 3 pedig az adott irányba történő elmozdulás mértékét.

6.2. Vezérléshez és adaptivitáshoz szükséges adatok küldése

A kliens alkalmazás TCP-n keresztül kommunikál a szerverrel. A TCP megbízható kommunikációt tesz lehetővé a kliens és a szerver között. Ez a TCP kapcsolódás és adat küldés JAVA nyelven kényelmesen kezelhető és az alábbi kódrészlet mutatja be:

```
InetAddress serverAddr = InetAddress.getByName(SERVERIP);
Socket socket = new Socket(serverAddr, SERVERPORT);
PrintWriter out = new PrintWriter( new BufferedWriter
            ( new OutputStreamWriter(socket.getOutputStream())),true);
out.println(control_value);
socket.close();
```

Ahol a *SERVERIP* jelen esetben a router IP címe, a *SERVERPORT* az engedélyezett port a felhasználónak és a control_value pedig az érték amit a megnyitott TCP socketen átküld a program.

WiFi kapcsolat paramétereinek lekérdezése

A WiFi kapcsolat állapotának lekérdezésére a legegyszerűbb módszer létrehozni egy *BroadcastReceiver-t*. Mely rendszerszintű eseményekre reagál. Ilyen rendszerszintű esemény a kapcsolat jelenlegi állapotának küldése is. Ezen értékek lekérdezése az alábbi módon történhet:

```
ConnectivityManager myConnManager = (ConnectivityManager)
getSystemService(CONNECTIVITY_SERVICE);
            NetworkInfo myNetworkInfo = myConnManager
                        .getNetworkInfo(ConnectivityManager.TYPE_WIFI);
            WifiManager myWifiManager = (WifiManager)
getSystemService(Context.WIFI_SERVICE);
            WifiInfo myWifiInfo = myWifiManager.getConnectionInfo();
```

A ConnectivityManager használatával ezek után a kapcsolat változó RSSI (received signal strength indicator [dBm]) értékeit és a link sebességét így lehet lekérdezni:

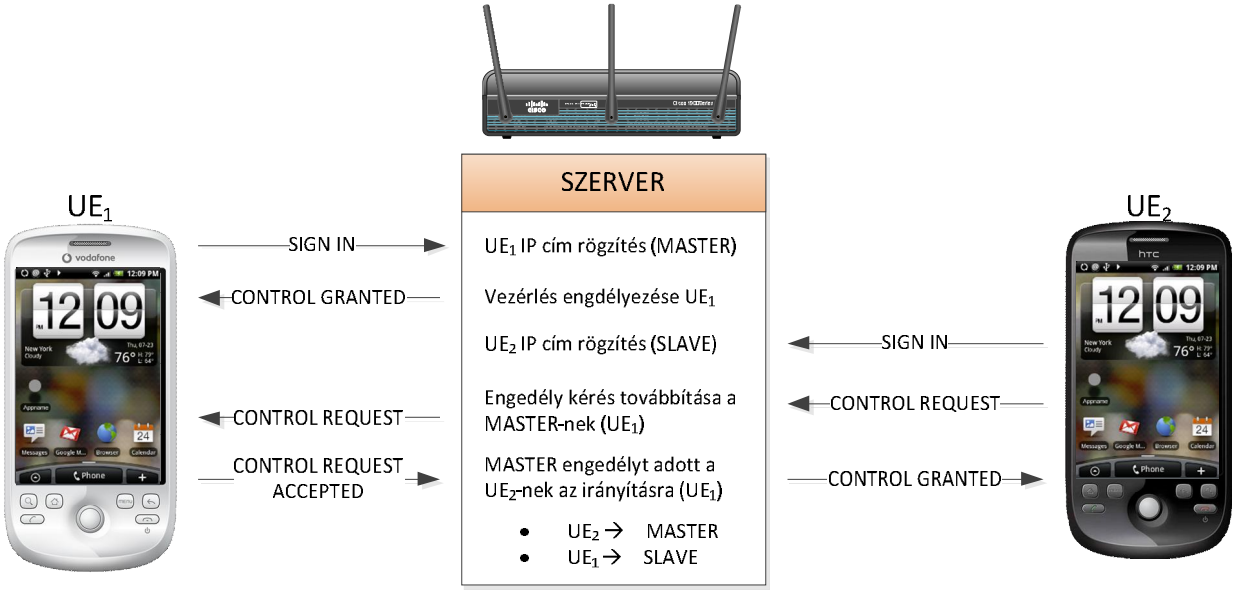
```
myWifiInfo.getLinkSpeed() //link sebessége

myWifiInfo.getRssi() //RSSI érték a felhasználónál mérve
```

6.3. Vezérlés átadása

A felhasználók és a szerver közötti üzenet váltások láthatóak a 6.3. ábrán. Az ábra egy olyan esetet mutat be amikor több kliens alkalmazás is csatlakozik a szerverhez, és egy felhasználó vezérelhet (MASTER - UE_1[3]). Ilyenkor a többi felhasználó nem kapja meg a *CONTROL GRANTED* jelet és így nem hívódik meg az ábrán az activity. Amennyiben UE_2 szeretne vezérelni ezt kérvényezheti a *CONTROL REQUEST* jellel. Ekkor a szerver alkalmazás megvizsgálja, hogy milyen IP cím van bejegyezve mint MASTER és erre az IP címre kiküld egy *CONTROL REQUEST* jelet, ami meghívja a 6. fejezetben ismertetett *ManageControl* osztályt. Ezután a MASTER eldöntheti, hogy engedélyezi a kérést vagy nem. Ha nem engedélyezte akkor elküldi a szervernek a *CONTROL REQUEST DENIED* parancsot és a szerver a megfelelő helyre továbbítja.

[3] User Equipment - felhasználói készülék



6.3. Vezérlés átadása másik felhasználónak

Engedélyezés esetén a *CONTROL REQUEST ACCEPTED* üzenetet küldi a MASTER a szervernek, ezzel egy időben már megszünteti a vezérlést ami a *AccelerometerManager.stopListening()* meghívásával történik. A szerver a kérvényezőnek eljuttatja a *CONTROL GRANTED* jelet és ekkor a UE_2 megkapta a vezérlés jogát. Természetesen ha csak egy felhasználó csatlakozik a szerverhez ő mindig megkapja a vezérlés jogát.

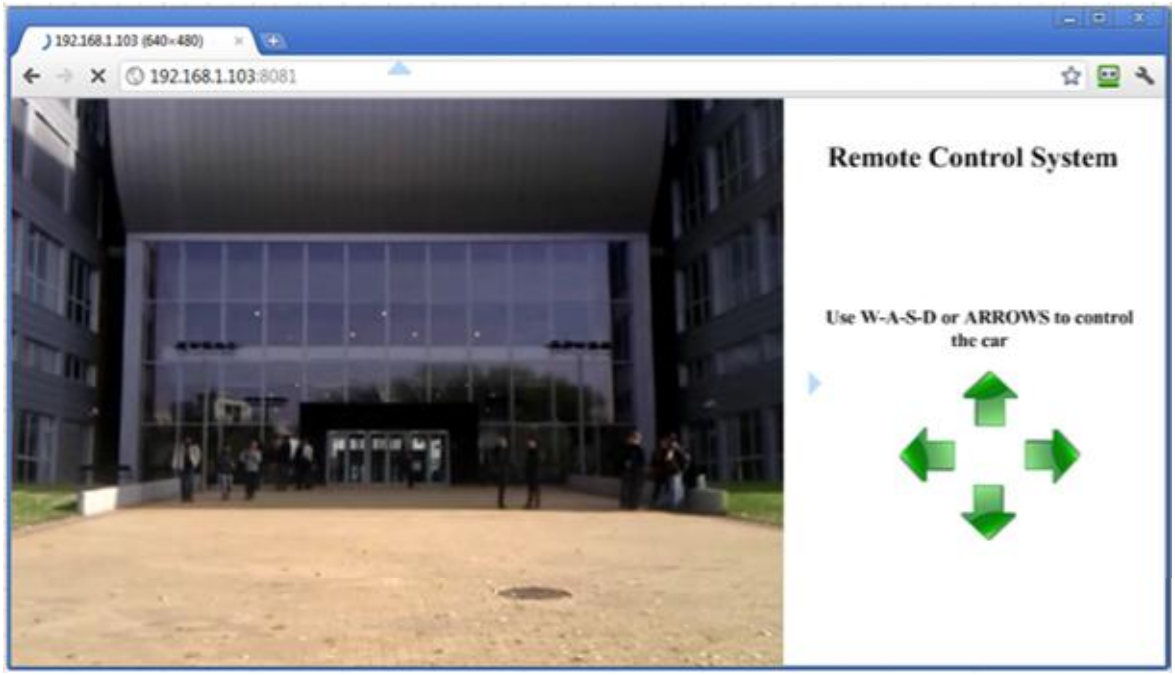
6.4. Android alapú kliens alkalmazás felhasználó felülete

A kliens alkalmazásnál a felhasználói felület megtervezésekor arra törekedtem, hogy minél nagyobb legyen a megjelenített valósidejű kép mérete, hiszen az okos telefonok kijelzőjének felbontása rohamosan növekszik, azonban a kijelző mérete így is korlátozott. Ezért a telefon teljes kijelzőjén a videót jelenítettem meg és nem volt szükség különböző nyilak feltüntetésére az irányításhoz, ami csak csökkentette volna a megjeleníthető kép méretét. Mivel a 6.1. fejezetben tárgyalt gyorsulásmérő szenzort alkalmaztam az irányításhoz. A 6.3. fejezetben részletesen tárgyalt vezérlés igény a menüből érhető el. Ezen felül a beérkező vezérlés átadás kérés esetén felugró ablak jelzi ezt a felhasználónak. A teljes képernyős felhasználó felület a **6.4**. ábrán látható (melyen a Q épület látható 320x480 pixel felbontásban).

6.4. ábra: androidos kliensalkalmazás felhasználói felülete

7. Platform független verzió felhasználói felülete

A rendszer fejlesztése során nem akartam csupán az Android platformra korlátozni a távvezérlést, ezért szükségesnek láttam egy a szerven futó egyszerű script implementálását, mely minden web böngészőből elérhető amely támogatja az MJPEG stream lejátszását (Firefox, Internet Explorer, Opera, Chrome, Konquerer stb.), így a vezérlés alap funkcionalitását biztosítani lehet platform függetlenül (Windows 7-en futó Chrome böngészőből a szerveroldali alkalmazás felhasználó felülete a 7.1. ábrán).



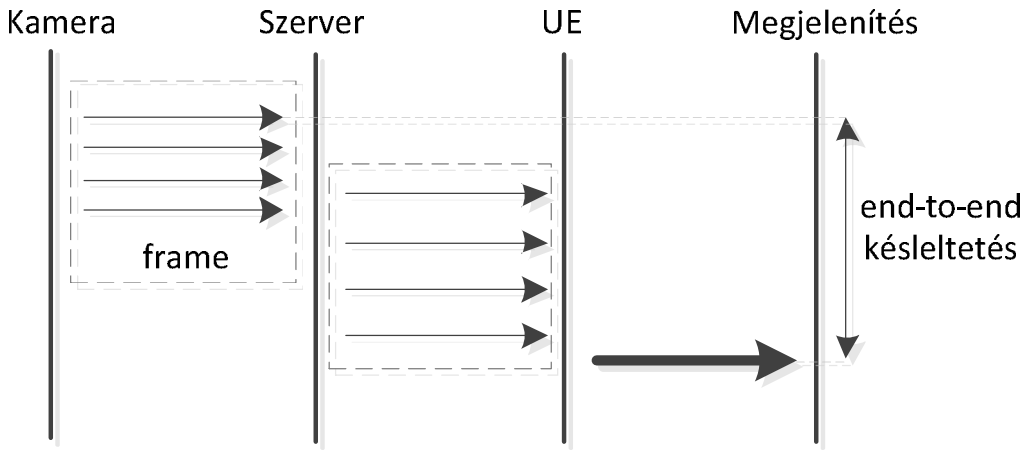
7.1. ábra: platform független vezérlés felhasználói felülete

8. Adaptív videó streaming

Valós idejű videónál nem lehet ugyanazokat az adaptációs megoldásokat alkalmazni, mint a tárolt videóknál, mert a valós idejű videó nem tud gyorsabban vagy lassabban "adni" a hálózatnak megfelelően, mint ahogy egy előre rögzített állománynál lehet nagyobb vagy kisebb rátával olvasni. Megoldás lehet, hogy playout buffer méretét a hálózati késleltetésekhez és jitterhez igazítani. A playout buffer hivatott a hálózaton késleltetett kép szekvenciák kijelzésére gyakorolt hatását minimalizálni. Ekkor azonban a bufferben eltöltött várakozással a valós idejű kép többlet késleltetést szenved. Ennél az alkalmazásnál, ahol a szolgáltatott valós idejű kép alapján történik az eszköz vezérlése, nem engedhető meg esetenként több másodperces késleltetés. Így egy másik megoldást kell megvizsgálni, ez a videó stream rátájának csökkentése, amit úgy érhető el, hogy az algoritmus az egyes JPEG képeket eltérő mértékben tömöríti az éppen szükséges bit rátára. Természetesen az egyes képek méretbeli azonossága így sem garantált, mivel azok tartalom függvényében különböző méretűek lehetnek. Élő videó képnél a legfontosabb minél alacsonyabban tartani a videó felvételétől a kijelzéséig eltelt időt (end-to-end késleltetés). Ez határozza meg, hogy mennyivel késleltetve jelenítődik meg a valós kép. Két fő oka van az end-to-end késleltetésnek: a felhasználói készülék és a hálózat. A vég-berendezések késleltetésébe beletartozik a küldő és a fogadó eszköz feldolgozási ideje és a bufferekben történő várakozás. Feltételezem, hogy a feldolgozási idők (tömörítés, dekódolás) egy felbontáson belül nem változik számottevően, azonban más-más felbontások esetén ezek lényegesen eltérő értékű lehet (nagyobb felbontású kép tömörítése és dekódolása több erőforrást vesz igénybe). Minél nagyobb egy kép mérete annál több MTU[4] (maximum transmission unit) méretű egységet használ fel (ha nem fér bele egybe). És ezeket mindegyiket meg kell, hogy kapja a felhasználó, hogy dekódolhasson egyetlen képet (8.1.ábra). Ha az átvitel minősége olyan mértékűre romlik, ami által a kapcsolat sebessége is a videó stream névleges rátája alá romlik, akkor a felhasználónál a playout bufferben kell várakozni a frame szegmenseknek mire az egy képhez tartozó összes szegmens megérkezik, és csak ezután kezdődhet meg a kép dekódolása. Ez megnövekedett végek közötti késleltetést jelent. Tovább súlyosbítja ezt a jelenséget, hogy a jel-zaj viszony romlásával nem egyenesen arányos a sebesség csökkenés az adott jel-zaj viszonyhoz szabvány által definiált modulációs és kódolási séma miatt [20]. Ezekre a moduláció váltásokra azért van szükség, mert a nagyobb jel-zaj viszonyhoz robosztusabb moduláció szükséges. Ez

[4] az MTU meghatározza bájtokban a legnagyobb átvihető egységet protokol szinten

alacsonyabb rátát eredményez, azonban szélesebb jel-zaj viszony tartományban teszi működőképessé a hálózatot. Ez a degradáció pontosan meghatározható a gyártói specifikációk és a 802.11 szabvány tanulmányozása alapján [20].

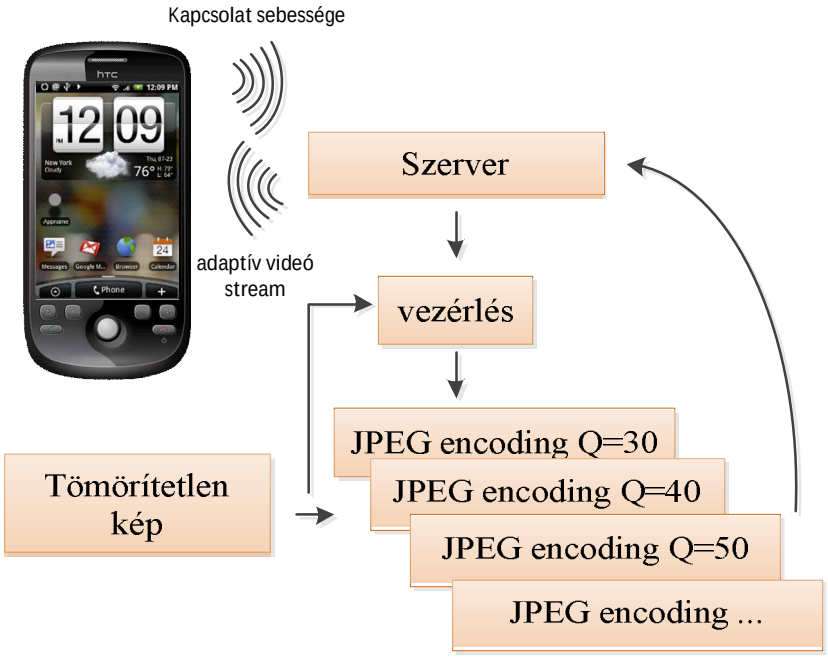


8.1. ábra: kép rögzítésétől a kijelzésig eltelt teljes idő

Ezek után megvizsgálom, hogy milyen paraméterek mellett tudja az adaptív algoritmus a legkisebb end-to-end késleltetést okozni a lehető legjobb képminőség mellett, és a valós rendszeren mérésekkel alátámasztom ezt.

8.1. Adaptív videó streaming algoritmus

Az adaptív videó streaming implementálásához visszacsatolást kell megvalósítani a felhasználói készülék és a szerver alkalmazás között. Ez a visszacsatolás a UE által küldött link sebesség értékek alapján valósul meg (8.2. ábra). A JPEG kódolásánál Q granularitását 10-re választottam és Q_{max}=80 és Q_{min}=30. Rosszabb minőségnél már látványos volt a kép romlása és tapasztalható volt az 5.4. fejezetben leírt blokkosodás is. Így összesen 6 féle "minőség" közül kell megválasztani a felhasználó számára optimálisat. Ez azt jelenti, hogy maximálisan 6 féle JPEG kódoló példányra van szükség, ha minden felhasználónak más-más képminőséget kell biztosítani. Amennyiben több felhasználónak megfelelő az azonos rátájú videó stream, úgy az csak egyszer állítódik elő és azt kapják meg a felhasználók.



8.2. ábra: visszacsatolás adaptív videó streaminghez

Első lépésben minden csatlakozott felhasználó elküldi az ő által végzett méréseket a kapcsolat sebességét illetően, ez a pszeudó kód első lépése is. A csatlakozott felhasználók száma $NUM_{connected\ users}$. A 2-3. lépésben az algoritmus becslést ad az éppen jelenlegi videó stream rátájának lehetséges értékeiről Q=30, 40, 50, 60, 70 és 80 JPEG paraméter mellett. Az 5. lépésben az első számú felhasználónál keresi meg a megfelelő Q értéket, ahol BW_i az i. felhasználónak becsült kapcsolati sebesség és $VR_{Q=20}$ pedig a videó becsült rátáját adja meg Q=20 esetén. Amennyiben megtalálta az algoritmus a megfelelő paramétert beállítja azt, ha nem addig folytatódik amíg nem találja meg.

1. kapcsolat sebességének BW_i meghatározása UE_i mérései alapján
2. előre meghatározott becslő mátrix az adott Q-hoz tartozó JPEG kompressziós arány
3. videó stream rátájának becslése a Q=100-hoz tartozó ráta a tömörítési arányok alapján
4. i=1;
5. **while** $i <= NUM_{connected\ users}$
6. **if** $BW_i > VR_{Q=20}$ and $BW_i < VR_{Q=40}$
7. $QA_i = 30$;
8. **else if** $BW_i > VR_{Q=30}$ and $BW_i < VR_{Q=50}$

9. $QA_i = 40$;
10. **else if** $BW_i > VR_{Q=40}$ and $BW_i < VR_{Q=60}$
11. $QA_i = 50$;
12. **else if** $BW_i > VR_{Q=50}$ and $BW_i < VR_{Q=70}$
13. $QA_i = 60$;
14. **else if** $BW_i > VR_{Q=60}$ and $BW_i < VR_{Q=80}$
15. $QA_i = 70$;
16. **else if** $BW_i > VR_{Q=70}$ and $BW_i < VR_{Q=90}$
17. $QA_i = 80$;
18. **end if**
19. **end while**
20. módosítani a JPEG kódolásnál használt Q paramétert UE_i-nél QA_i -re

9. Teljesítményelemzés

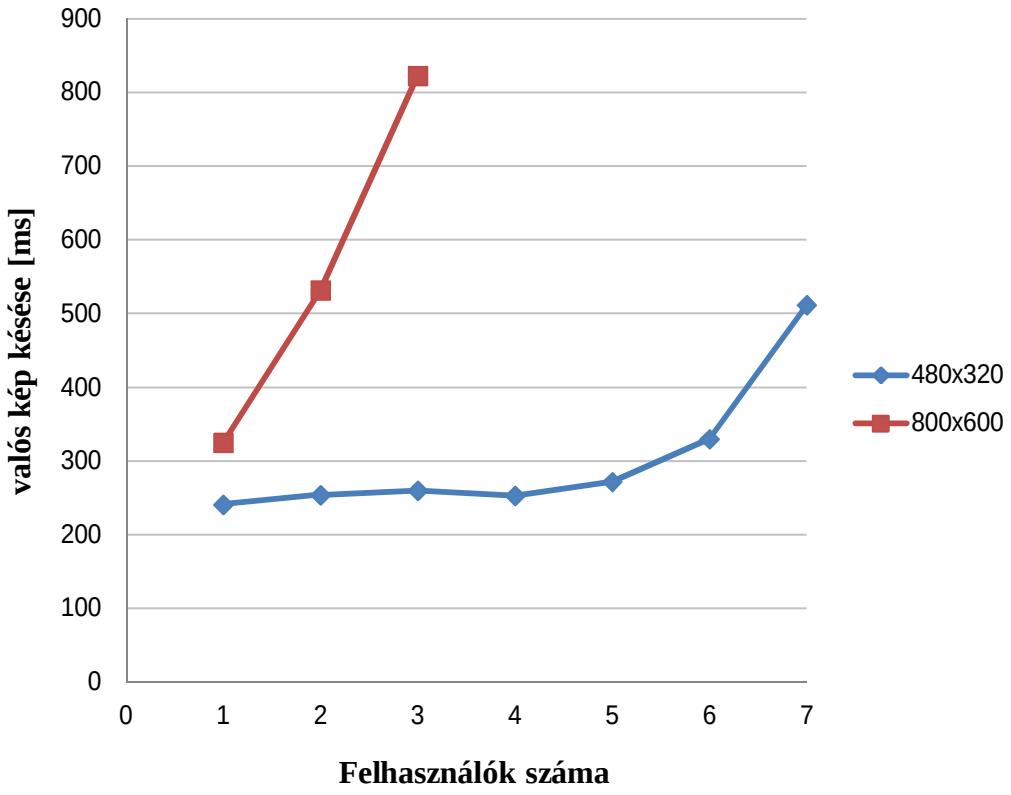
Az algoritmus és a működés igazolására több szempontból is elemeztem és lemértem a rendszer teljesítményét, melyek a következő alfejezetekben kerülnek ismertetésre.

9.1. Szerver stressz teszt

Mivel a szerver oldali alkalmazás képes több felhasználót kiszolgálni, ezért fontos a szerver teljesítményét megvizsgálni. Ezt a mérést úgy terveztem, hogy maximális terhelést okozzon a szervernek. Ugyanis a webkamera által támogatott legnagyobb felbontást vettem alapul ami 800x600 pixel, ezen felül megvizsgáltam az okos telefonoknál leginkább elterjedt HVGA felbontást is (480x320). Továbbá minden felhasználónak más-más rátájú MJPEG streamet osztottam ki ezáltal maximalizálva a szervernél a szükséges kép kódolási mennyisséget. A felhasználóknak véletlenszerűen sorsoltam MJPEG minőséget a Q=30-tól egészen a Q=80-ig. A forgalom generálására a VLC media player [21] konzolt alkalmaztam. Így könnyen automatizálható és jól skálázható mesterséges forgalmat sikerült generálnom. A 9.1. táblázat és a 9.1. ábra tartalmazza a tesztelés során mért eredményeket.

Felbontás [pixel]	Felhasználók száma	Átlagos késleltetés a UE-nál [ms]	CPU használat [%]
800x600	1	325	41
	2	532	93
	3	823	100
320x480	1	241	32
	2	254	45
	3	260	57
	4	253	64
	5	272	80
	6	330	95
	7	512	100

9.1. táblázat: a szerver teljesítmény vizsgálata



9.1. ábra: szerver teljesítmény vizsgálata felhasználó szám és késleltetés

A szerver teljesítményelemezése rámutatott, hogy a 800x600 pixel felbontás esetén már 3 felhasználó esetén a valós kép késleltetése már kiugróan magas lett, így 3 felhasználó esetén már nyújtotta az elvárt szolgáltatási minőséget. Megvizsgáltam, hogy 2 felhasználó esetén, mi

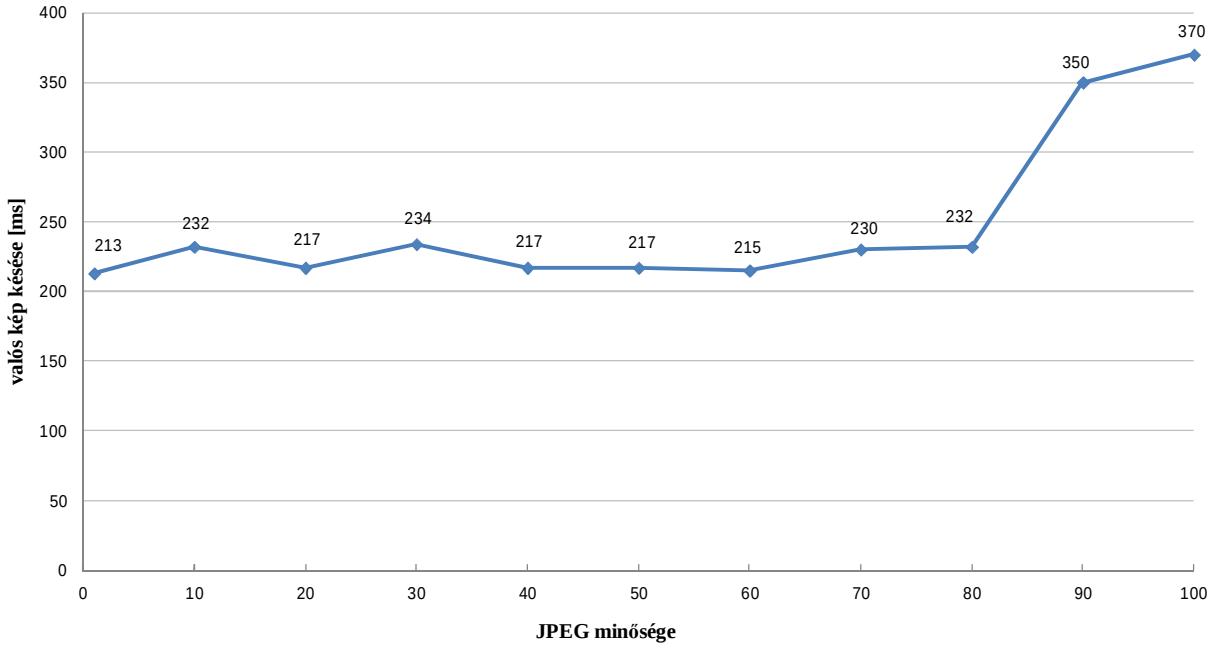
indokolta az ilyen mértékben megnövekedett valós idejű kép késleltetését. Ezért megmértem a routernél a WiFi kártya maximum átviteli sebességét a széles körben alkalmazott IPerf [22] programmal. Ezek a mérések azt mutatták, hogy az elérhető sávszélesség ~28 Mbps volt. Ami magas rátájú videó streamnél a szűk keresztmetszetet okozta. A másik vizsgált felbontásnál már jóval több felhasználót ki tudott szolgálni a szerver és a szolgáltatás minősége sem tért nagyban el a 9.2. fejezetben meghatározott késleltetési minimum értéktől. 6 felhasználót még átlagosan 330 ms késleltetésű képpel ki tudta szolgálni a szerver 95 %-os CPU terhelés mellett. Ez igen figyelemre méltó eredmény, hiszen minden felhasználó megkapta az alacsony késleltetésű 320x480 pixel felbontású képet.

9.2. Az adaptív algoritmus vizsgálata

Az implementált adaptív algoritmus vizsgálatához kidolgoztam egy mérési módszert a valós rendszeren. Azt kell tudni modellezni, hogy a felhasználónál romlik a jel-zaj viszony, és erre az adaptív algoritmus ezeket az információkat is felhasználva megváltoztatja a videó stream rátáját a kapcsolat átviteli sebességéhez mérten. Ebben a mérésben az a nehézség, hogy pontosan be kell tudni állítani a felhasználónál a mérhető jel-zaj viszonyt, ami például a távolság változtatásával a router és az UE között változtatható. Azonban ez esetenként több száz métert is jelenthet, ami a mérést nagyban pontatlanná és nehezen megismételhetővé tenné. Ezért a routeren a rádiós interfész sugárzási teljesítményét változtatom, ezzel szimulálva a felhasználó "távolodását". Ezt a változtatást szoftveresen el lehetett végezni, így a mérés jól automatizálhatóvá vált. A teljes késleltetést úgy lehetne mérni a legteljesebben, hogy például egy stoppert nézni a kamerával és ezt összehasonlítani a felhasználónál megjelenített képpel. Ekkor a két órán eltelt idő a rendszer teljes késleltetése a feldolgozási és továbbítási időket is beleszámolva. A 9.2. ábrán egy ilyen mérés figyelhető meg: a bal oldali óra a kamera által vett kép a szerver oldalon (másodperc és századmásodperc) megjelenítve a felhasználónál, a jobb oldali pedig a kijelzés pillanatában éppen látható idő. Így a két kijelzett idő különbsége megadja a teljes késleltetést. Természetesen itt is adódnak bizonyos pontatlanságok. Például a kamera által egy másodperc alatt rögzíthető képek száma miatt, vagy a leolvasási pontatlanságok az érték váltások alkalmával.

9.2. ábra: a valós kép teljes késleltetésének egy lehetséges mérése

Ez a módszer a gyakorlatban több automatizált mérés elvégzésére nem alkalmas, ezért úgy jártam el, hogy a kép felvételének időpontját rögzítettem minden képnek és azt az időbélyeget vettem amikor az összerakott kép elhagyta a playout buffert a felhasználónál mérve.
A következőkben megvizsgálom, hogy milyen hatással van a különböző mértékben tömörített kép a valós idejű kép késleltetésére. A méréshez felhasznált videó stream felbontása 800x600 pixel. A legjobb minőségnél 9 bit/pixel alkalmazása mellett [17] körülbelül 30 Mbit/s szükséges. Ez a ráta a legtöbb mobil készüléknek melyek WiFi interfészének névleges sebessége 54 Mbps, a tényleges sebességénél túl mutat. Ezért megnövekedett késleltetésre kell számítani. A teszteléshez használt készüléknél (HTC WildefireS) a mért tényleges adatátviteli sebesség 22 Mbps körül alakult (a névleges 54 Mbps). A következő mérést tökéletes vételi körülmények mellett mértem. Minden mérésnél 5 másodpercnyi (150 kép) kép átlagából képeztem az eredményeket.

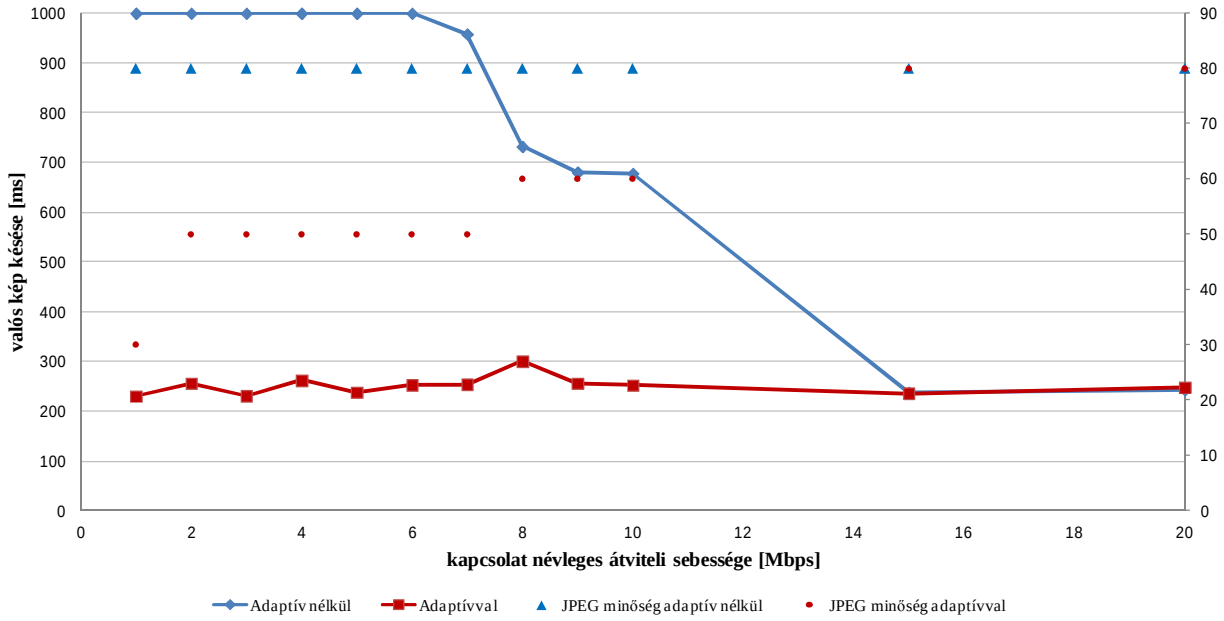


9.3. ábra: a valós idejű kép késleltetése a JPEG kép minőségének függvényében

A 9.3. ábrán láthatóak a mérési eredmények. Megfigyelhető, hogy a 80-as minőség alatt már nem vehető észre lényeges különbség a késleltetések között. Ez azzal magyarázható, hogy például a JPEG Q=80 mellett már a videó stream rátája ~15 Mbps-ra csökkent. Így a hálózati késleltetések nem adnak többlet késleltetést a valós kép késleltetésének, még a Q=1 esetén is ez a mérhető, pedig ott a videó stream rátája körülbelül 0.2 Mbps. Tehát az itt mért késleltetés a minimális késleltetés, ami a teszt rendszernél mérhető. Ez a késleltetés lesz az adaptív videó stream algoritmus vizsgálatakor az elérni kívánt cél a lehető legjobb kép minőség mellett.

A következő mérésben megvizsgáltam az adaptív videó streaming algoritmus hatékonyságát, ahhoz képest amikor nem alkalmaztam ilyet, hanem végig egy előre meghatározott rátával történt a valós kép küldése. A 9.4. ábrán látható a mérési eredmény, a piros jelölő (négyzet a késleltetés, kör pedig a minőség) az adaptív videó streaminghez tartozó eredmények, a kék (kör a késleltetés, háromszög a stream minősége) pedig az fix videó streaminghez tartozó eredmények. A videó stream fogadásának és megjelenítésének implementálása során a playout buffer méretét 1 másodpercben maximalizáltam, tehát ha ennél többet kell várakozni a frame szegmenseknek a felhasználónál akkor eldobódnak. A fix videó stream esetében látható, hogy 6 Mbps alatti kapcsolat sebesség alatt mindig volt eldobódott frame a playout buffer miatt. Ezzel szemben az adaptív megoldásnál a késleltetés mértéke azon a szinten mozgott, mint amit meghatároztam a rendszer minimális késleltetésének (~230 ms).

Látható az adaptív algoritmus a kép minőségét Q=80 és Q=30 között változtatta. Ezáltal nem a videó stream rátájának nem megfelelő megválasztása nem okozott többlet késleltetést a hálózatban.



9.4. ábra: az adaptív algoritmus hatása a valós idejű kép késleltetésére

10. Eredmények értékelés

Az implementált rendszer képes volt azokat a szolgáltatásokat és elvárásokat nyújtani, amik a tervezéskor elérni kívánt cél ként megfogalmazódtak. Ezek többek között a kliensalkalmazás megvalósítása, mellyel lehet táv vezérelni az eszközt WiFi-n vagy beépített adatkapcsolaton keresztül és képes valós idejű videó stream megjelenítésére. Továbbá az adaptív videó streaming algoritmus megvalósítása, kifejezetten alacsony késleltetésű videóra optimalizálva. A szerver alkalmazás megvalósítása ami képes vezérelni a teljes folyamatot. Végül az elektronika megtervezése és megvalósítása, melynek segítségével megvalósítható a periféria illesztés és vezérlés a szerverhez.

A részletes teljesítményelemzés megmutatta, hogy megfelelő robosztusságot biztosít a szerver oldali szoftver és hardver egyaránt. Képes volt egyidejűleg 6 felhasználó kiszolgálására (320x480 pixel felbontású videó esetén) anélkül, hogy a szolgáltatás minősége a felhasználó által észrevehető mértékben romlott volna. Ez a beágyazott szervertől figyelemre méltó teljesítmény és az alkalmazás jellege nem is igényelne nagyobb teljesítményt.

A megvalósított adaptív algoritmus teszteléséhez kidolgoztam egy jól automatizálható mérést a valós rendszeren. A mérések során bebizonyosodott, hogy az adaptív videó streaming algoritmus minden vizsgált esetben hatékonyabb a valós idejű kép továbbítására, mint a fix videó stream rátájú megoldás. Amikor a szűk kapacitás a sávszélesség volt, akkor a megoldások között 800 ms-mal kisebb késleltetést lehetett elérni az adaptív algoritmussal. Ami a vezérlésnél döntő fontosságú lehet, hiszen majdnem egy másodperc alatt akár több tíz métert is haladhat "vakon" az irányított eszköz.

11. Összefoglalás és továbbfejlesztési lehetőségek

Távvezérlésű eszközöket széles körben alkalmaznak a legkülönfélébb feladatokra. Az okos telefonok és a tabletek nagy mértékű elterjedésével, egyre nagyobb az igény, hogy a vezérlések irányítását és ellenőrzését ilyen készülékekről lehessen irányítani.

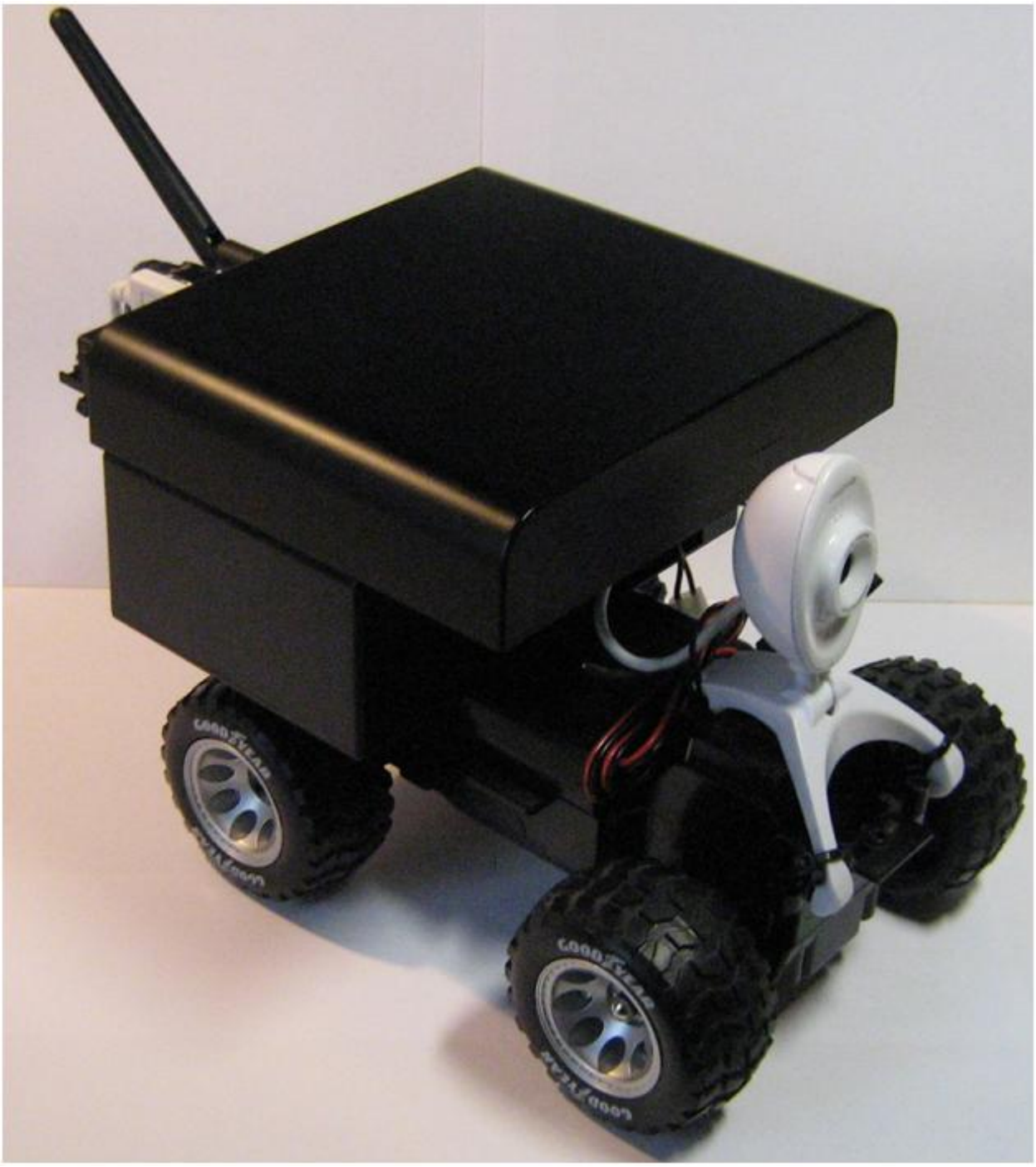
A TDK dolgozatban egy továbbfejlesztett távvezérlő rendszert terveztem és valósítottam meg. Munkám során számos újszerű problémába ütköztem mind a hardver, mind pedig a szoftver tervezése és készítése során, melyek megoldására a dolgozat egyes alfejezeteiben részletesen kitértem. A vezérlés komponenseit úgy terveztem meg, hogy minél szélesebb körben, rugalmasan lehessen más alkalmazásnál is használni. Mivel ez egy olyan rendszer ami ipari környezetben is alkalmazható, ezért teljes körű teljesítmény elemzést végeztem mind a szerver által kiszolgálható kliensek számát illetően, mind pedig az így nyújtott szolgáltatás minőségét.

A szakirodalomban fellelt megoldások tanulmányozása során felmerült, hogy sávszélesség szűkös esetben a fix rátájú videó streaming megoldások nem nyújtanak kellően alacsony késleltetésű valós képet. Így szükségessé vált egy olyan adaptív streaming algoritmus megvalósítása, mellyel kisebb rendelkezésre álló sávszélesség esetén is minimális késleltetésű kép biztosítható a felhasználó számára.

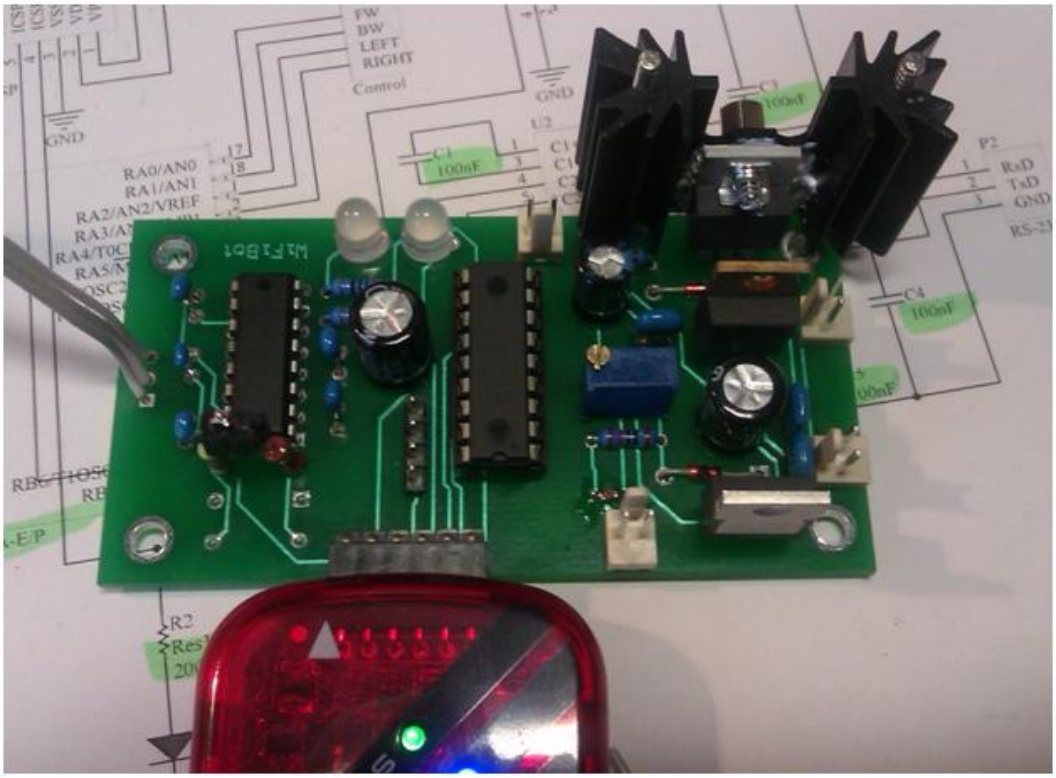
Mivel a videó stream-hez MJPEG folyamot használtam, így az interframe redundancia miatt nagyobb sávszélességet kellet használni egyéb ezt kiszűrő videó tömörítési eljárásokhoz képest.

Érdemes lehet olyan megoldást is megvizsgálni ahol az MJPEG alkalmazása helyett más a hálózati terhelés szempontjából hatékonyabb megoldás például MPEG-4 videó stream kerül alkalmazásra. Az adaptív algoritmus jelenleg a kapcsolat sebességét és a jel-zaj viszonyt vette figyelembe, tehát egy másik továbbfejlesztési irány lehet az adaptivitás kiterjesztése a hálózati késleltetések és csomagvesztések minimalizálására is.

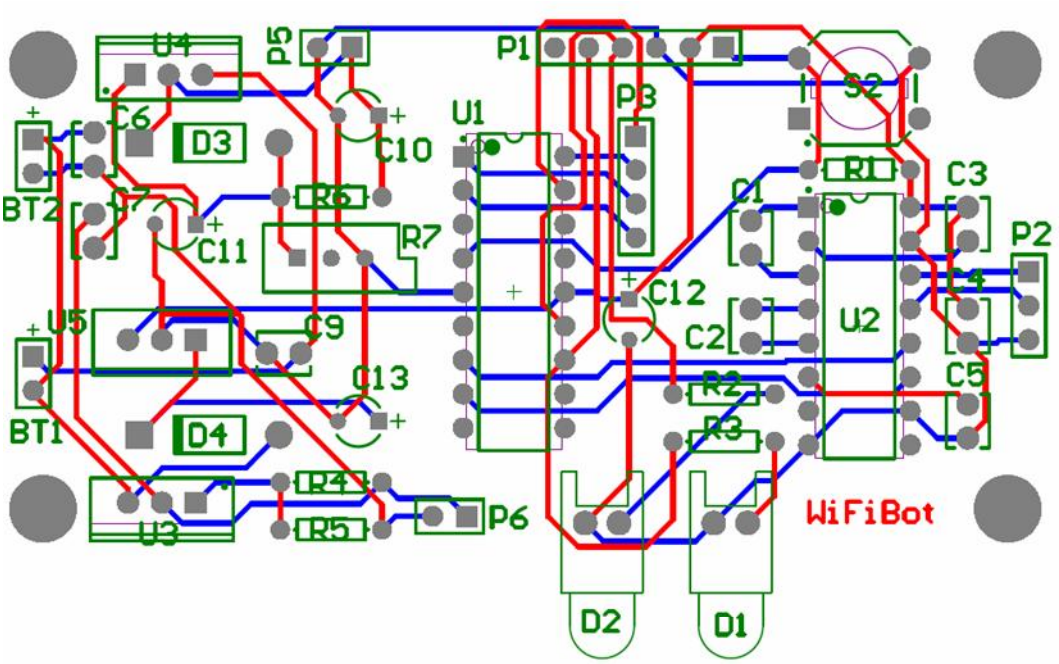
12. Melléklet



12.1. ábra: a megvalósított távvezérlésű modell autó



12.2.ábra: a vezérlő elektronika élesztés közben



12.3. ábra: a vezérlő elektronika huzalozási terve

13. Irodalomjegyzék

[1] Google. (2011) Android@Home. [Online]. http://www.androidathome.com/ , 2011 október

[2] Parrot SA. (2011) Parrot AR.Drone. [Online]. http://ardrone.parrot.com/parrot-ar-drone/uk/ , 2011 október

[3] Yaodong, Wenbin, "Internet-based tele-control system for wheeled mobile robot," , 2005, pp. 1151 - 1156.

[4] Contet, Ruichek Nogueira. (2007) Wifi based remote control system with video feedback for intelligent vehicles.

[5] Hayashida, Hayashi Nishantha, "Application level rate adaptive motion-JPEG transmission for medical collaboration systems," , 2004, pp. 64 - 69.

[6] Seung-Chur, Jong-Deok Se-Mi. (2011) Design and Implementation of a Smartphone-Based Reliable Real-Time Wi-Fi Broadcast System. [Online]. Design and Implementation of a Smartphone-Based Reliable Real-Time Wi-Fi Broadcast System

[7] Yang, Song Chang. (2011) A Scalable Mobile Live Video Streaming System Based on RTMP and HTTP Transmissions. [Online]. http://www.springerlink.com/content/g4831650g1211282/

[8] Venkataraman, Muntean Kennedy. (2011) Dynamic stream control for energy efficient video streaming. [Online]. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5954971&tag=1

[9] OpenWrt. (2011, Oct.) OpenWrt. [Online]. https://openwrt.org/ , 2011 október

[10] Microchip. (2006) Microchip. [Online]. http://ww1.microchip.com/downloads/en/DeviceDoc/40300C.pdf

[11] MAXIM. (2002) MAX232 DATASHEET. [Online]. http://www.datasheetcatalog.org/datasheet/texasinstruments/max232.pdf

[12] Giovanni Giacobbi. (2011, Oct.) Netcat. [Online]. http://netcat.sourceforge.net/ , 2011 október

[13] (2005) How to use serial port to communicate between two computers. [Online]. http://www.codeproject.com/KB/IP/serialporttocommunicate.aspx , 2011 október

[14] Andersen. (2011, oktróber) BuildRoot. [Online]. http://buildroot.uclibc.org/ , 2011 október

[15] Steinbach, Farber Girod, "Adaptive playout for low latency video streaming," in *IEEE*, 2001, pp. 962-965.

[16] Stoeveken. (2010) Mjpg-streamer. [Online]. http://sourceforge.net/projects/mjpg-streamer/ , 2011 október

[17] Joint Photographic Experts Group. (2011, Oct.) JPEG standard. [Online]. http://www.itu.int/rec/T-REC-T.81-199209-I/en

[18] FFmpeg. (2011, Oct.) FFmpeg. [Online]. http://ffmpeg.org/index.html , 2011 október

[19] Google. (2011) Android developers. [Online]. http://developer.android.com/index.html , 2011 október

[20] IEEE. (2011, Oct.) Wireless Local Area Networks. [Online]. http://standards.ieee.org/about/get/802/802.11.html

[21] VideoLAN. (2011) VLC. [Online]. http://www.videolan.org/vlc/ , 2011 október

[22] Iperf. (2011) Iperf. [Online]. http://iperf.sourceforge.net/ , 2011 október

14. Ábrajegyzék