Okostelefonnal, WiFi-n keresztül vezérelt modellvasút

Kovács Tamás okl. villamos üzemmérnök, ERICSSON Hungary

Az itt bemutatni kívánt alkalmazás játéknak is tűnhet első látásra, azonban a valódi cél egy korszerű demó alkalmazás bemutatása volt, amely modern vezérléstechnikai és informatikai eszközöket alkalmaz és rámutat azok hatékony kihasználására, akár hobby célokra is.

Lássuk mit is kínál ez az alkalmazás. Célunk egy modellvasút mozdonyainak irányítása mobil eszközről (tablet, mobiltelefon) WiFi kapcsolaton keresztül, mobilapplikáció felhasználásával. A mobilapplikációtól elvárjuk, hogy az irányítandó mozdonyok koppintással egy listából kiválaszthatók legyenek, minden paraméterük adminisztrálható legyen. A mobilapplikációból a mozdonyok haladási iránya, sebessége változtatható legyen, a modell valósághűen fékezzen le, ill. induljon el, de vészmegállásra is legyen lehetőség. A mozdonyon levő menetfények a haladási iránynak megfelelően automatikusan működjenek, előre fehér fény, hátrafele piros zárfény világítással. Mivel a mozdony vezérlése teljes egészében rádiós úton történik, a sín csak a tápfeszültséget szolgáltatja, azon keresztüli vezérlőjelek nem befolyásolják, így meglévő digitális pl. DCC, Locopilot rendszer kiegészítője is lehet.

Az egész rendszer úgy lett kialakítva, hogy minden hardver és szoftver komponense más célokra is felhasználási mintaként szolgáljon.

A vasútmodellezéssel kapcsolatban elmondható, hogy a régebbi, analóg modellek digitalizálása igen költséges, és munkaigényes feladat. Ezzel a megoldással, a gyári rendszerek költségének töredékéért, azzal megegyező, némely vonatkozásban még azt meg is haladó színvonalú rendszert alakíthatunk ki.

Előnyök és hátrányok a hagyományos digitális modellvasút pl. DCC rendszerrel összehasonlítva:

Előnyök:

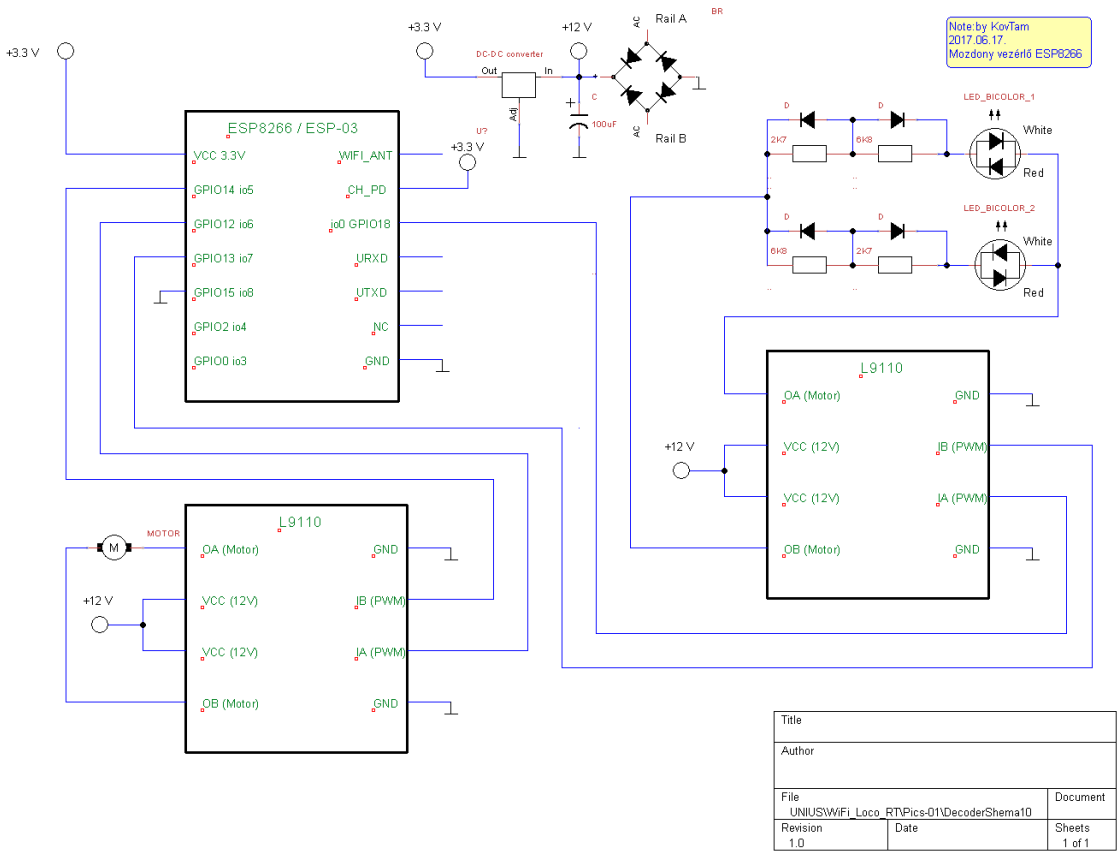
- Nincs szükség külön vezérlőközpontra.
- Nincs szükség a sínen keresztül, forgó kerekeken áthaladó, bizonytalan adatátvitelre.
- A sín csak tápfeszültség forrás.
- Olcsó megvalósíthatóság. (Az ESP8266-03-as vezérlőpanel ára kb. 3 USD, H bridge IC kb. 1 USD, RW Duo LED 0,5 USD)

Hátrányok:

- Viszonylag nagy áramfelvétel és hődisszipáció.

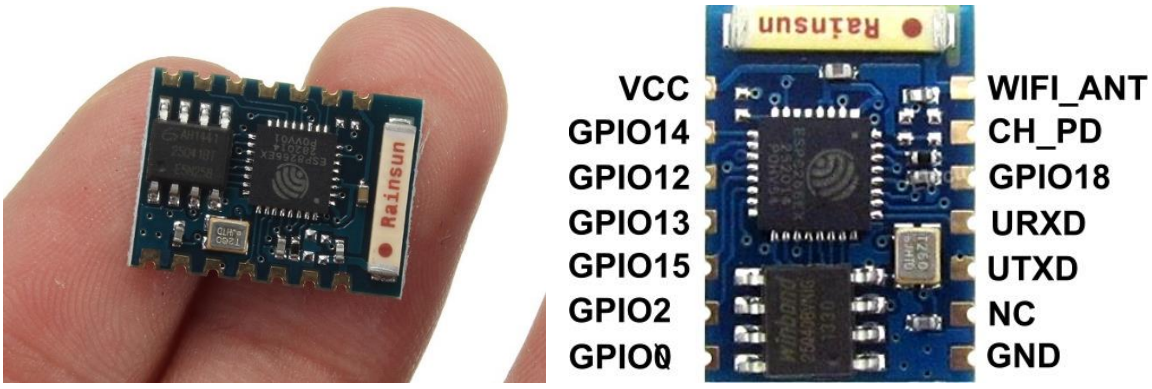
Rendszerjellemzők:

A modell vezérlése, a mozdonyban elhelyezett WiFi integrált vezérlőmodul, ESP8266, segítségével történik. A mozdonyban található, 3 vagy 5 pólusú, egyenáramú, állandómágneses szénkefés motor fordulatszám szabályozása impulzusszélesség modulációval PWM, forgásirány váltása pedig polaritás fordítás révén valósul meg. A motorvezérlő áramkör szintén integrált MOSFET H-hidat tartalmazó IC, Típusa: I9110. Az egész mozdonyvezérlő kapcsolási rajza az 1. számú ábrán látható.



1.ábra

Mint az előbbiekből is látható a vezérlés kulcs eleme az ESP8266 integrált WiFi chipet tartalmazó panel. Vizsgáljuk meg közelebbről ezt, a tényleg jól használható eszközt. Ez gyakorlatilag nem más, mint egy ultra miniatűr SBC (Single Board Computer), amelyen egy ún. lightweight RTOS operációs rendszer fut, azon felül pedig egy olyan szoftver csomag, amely egyszerű hozzáférési lehetőséget biztosít számunkra magasszintű szkript nyelven programozni az eszközt. Az itt bemutatott alkalmazásunkban a Nodemcu fejlesztésű lua szkript nyelvet fogjuk használni, annak okán, hogy ez talán a legegyszerűbb és a legkönnyebben használható, illetve interneten nagyon jól dokumentált, nyílt forráskódú és ingyenes.

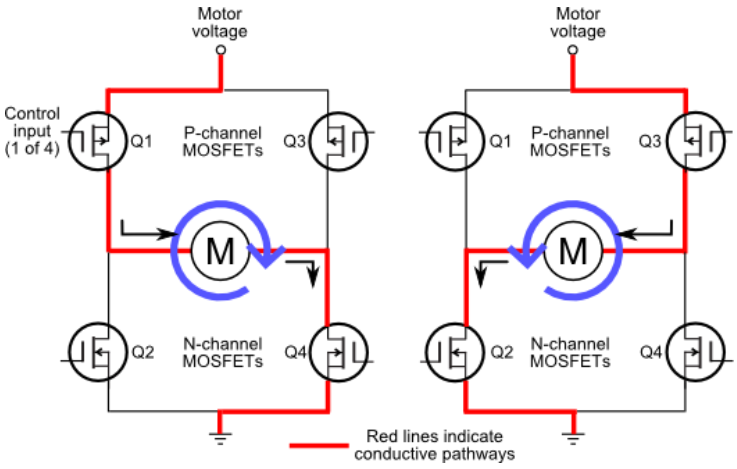


2.ábra

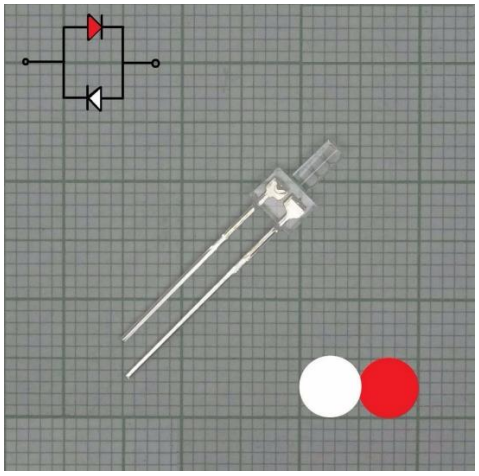
Az ESP8266-os chipet tartalmazó modulok, különféle kivitelben állnak rendelkezésre mind méretben, mint pedig a szerint, hogy hány lábat vezettek ki róla felhasználás igényeinek megfelelően. A mi

esetünkben a választás a ESP-03 típusra esett, mivel az alkalmazásunk a kötelező perifériák mellett összesen 4db GPIO (General Purpose Input Output / Általános felhasználású kimenet vagy bemenet) kimenetet igényel. A panel mérete mindössze 17x12 mm, 2. ábra. Kicsi a bors, de erős. Kis méretéhez képest tartalmazza az ESP8266 chip-et, amely magában foglal egy 32 bites CPU-t 160 MHz órajellel, 128 Mbyte RAM-ot, komplett Wifi 802.11 b/g/n rádiós frontend-et, periféria interfészeket. 2 Mbyte SPI FLASH-t a programok tárolására. A panelen integrált WiFi antenna is helyet kapott. A ESP8266 CPU számos, szabványos periféria interfészt támogat, pl. A/D átalakító számára dedikált port-ot, I2C, SPI, 1-Wire busz csatlakozási pontokat, és két USART (RS-232) port-ot is. A mi alkalmazásunk ezekből csak az elsődleges USART port-ot és 4 digitális GPIO lábat használ. A USART port-ot is csak, a CPU szoftver feltöltéséhez használjuk.

A mozdonyvezérlő áramkör működése röviden: A mozdony villanymotorját egy ún. H-híd áramkör segítségével vezéreljük. A H-híd működését a 3. ábra szemlélteti. A H-híd áramköri elemeit az l9110 IC kompletten tartalmazza. Itt csak a működés alapelvét mutatjuk be. Ammennyiben a Q1 és Q4 FET van nyitva, (Az l9110 'IA' bemenete van logikai H szinten) a motor az óramutató járásával megegyező irányba forog, ha pedig a Q3 és Q2 FET van nyitva, (Az l9110 'IB' bemenete van logikai H szinten) a motor az óramutató járásával ellenkező irányba forog. Ezzel a mozdony menetirányának változtatása megoldható,



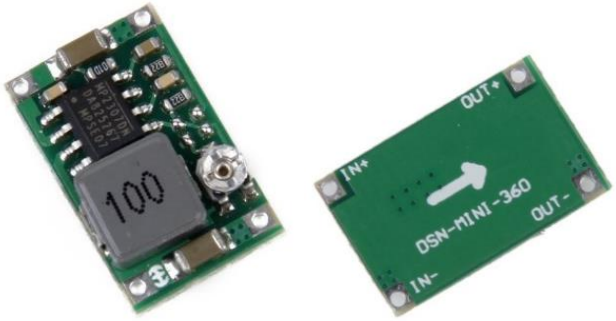
3. ábra



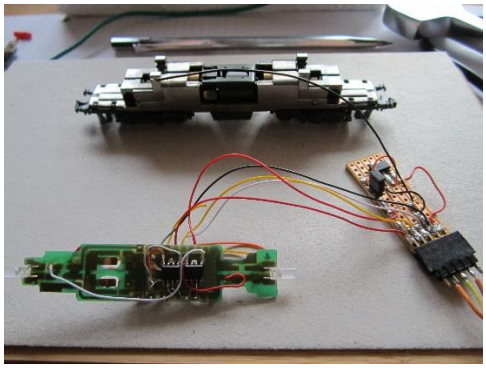
4. ábra

azonban, hogy a sebességét, azaz a motor fordulatszámát is szabályozni tudjuk, a párokban működő MOSFET-ek vezérlése impulzus szélesség modulált (PWM Pulse Width Modulation) jellel történik. A motor fordulatszáma, a PWM jel kitöltési tényezőjével együtt változik. A megfelelő kitöltési tényezőjű négyszög jelet a CPU-val szoftveres úton állítjuk elő a hídáramkör számára. A menetfényt egy speciálisan erre tervezett kétszínű fehér/piros LED-el oldjuk meg. 4.ábra. A menetirányban a mozdony fehér fénnyel világít előre, hátul az ún. zárfény pedig piros. A menetirány megváltozásával ez automatikusan felcserélődik. A LED fényváltása polaritás felcserélésével realizálható. Azonban biztosítani kell, hogy a kétféle színhez tartozó anti paralel kapcsolt dióda különböző áramot is igényel a működéséhez, ezért nem elég csak megfordítani a feszültség polaritását, a LED-en, hanem az 1. ábrán látható diódás söntáramkörrel is ki kell egészíteni. Mivel itt is polaritásfordítást kell alkalmazni, itt is H-hidat használunk, de itt nincs szükség PWM vezérlésre.

A mozdony, a tápfeszültséget a sínről, csúszó érintkezők által szedi le, ez analóg üzem esetén közvetlenül a meghajtómotorhoz van odavezetve. Ezt a kötést az átalakítás során bontani kell, és a sínből érkező tápfeszt a vezérlőpanel Graetz dióda hídjára kell csatlakoztatni, így biztosítva a polaritásfüggetlen tápellátás lehetőségét. Az ESP-03-as vezérlő panel 3.3 V-os tápfeszültségét, kapcsolóüzemű DC-DC Step Down konverterrel állítjuk elő típusa: DSN-mini-360, előnye, hogy nagyon kis méretű, mindössze 17x11 mm, hatásfoka 95%, olcsó, ára kb. 1 USD. 5.ábra. Az IN+ bemenetére kerül a Graetz egyenirányítóról lejövő szűrt, kb. 12V-os egyenfeszültség. Az IN- és az OUT- egyaránt testre kerül. Az OUT+ -on kapjuk a lekonvertált feszültséget, melyet a miniatűr pótméterrel 3.3 V-ra kell beállítani a mozdonyban.



5.ábra



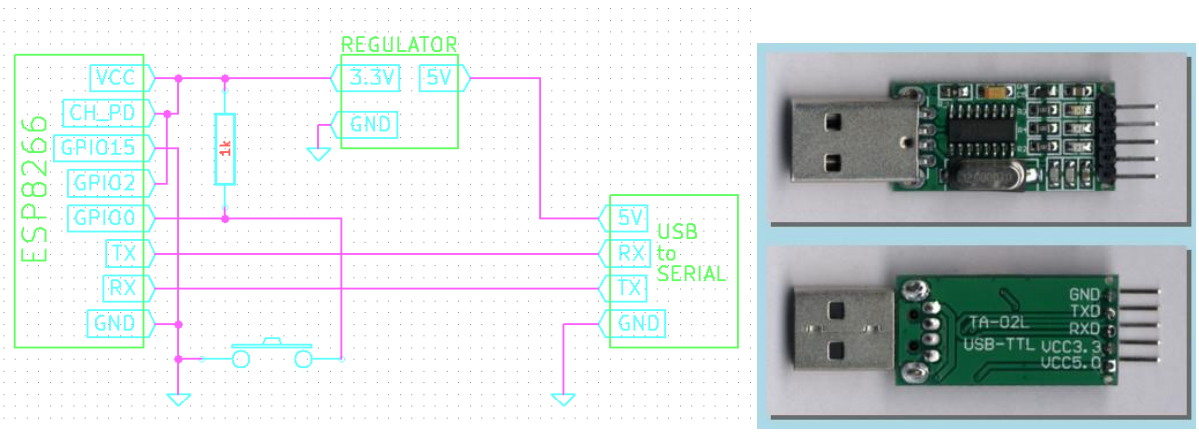
6.a ábra

A mozdony hardver kialakításához természetesen alkalmazkodni kell az adott modell típusához, az adódó méretekhez stb. Az itt bemutatott mintapéldányban gyakorlatilag minden légszerelten van összekötve. Az eredeti alaplapot leszámítva, egyetlen maratott NYÁK-ot tartalmaz, ami a két H híd IC-t tartalmazza. A bemutatott mozdony modell N méretarányú (1:160), tehát extrém kis méretekről van szó. Egy H0-as, vagy TT-s mozdonynál jóval könnyebb dolgunk lehet. A mozdony egy HobbyTrain MÁV Taurus, áramköri kialakításáról néhány képet szeretnék bemutatni, ötletadónak 6.ábra. A panelek szerelésénél vegyük figyelembe, hogy azok, amennyire lehetséges jól átszellőző helyre kerüljenek, hogy kerüljük a túlmelegedést.

6.b ábra

A mozdony szoftverének feltöltéséhez szükség lesz egy ún. USB-TTL átalakítóra és némi segéd vezetékezésre is. Az ehhez szükséges USB-TTL átalakítót és az ideiglenes kapcsolást a 7.ábra mutatja.

Fontos! Az ESP8266EX chippel szerelt WiFi modul 3.3V tápfeszültséget igényel, és az 5V tápfeszültség azonnal károsítja! GPIO bemenetei azonban 5V toleránsak, de azért célszerű betartani ezeknél is 3.3V-os logikai H szintet. Az USB-TTL konverter is legyen 3.3V-os, általában ezek rendelkeznek 3.3V-os táp kimenettel is az 5V-os mellett. Javasolt azonban a segédkapcsoláson is bemutatott módon az 5V-os kimenetre egy 3.3V-os regulátort tenni és arról táplálni a modult a programozás alatt. Az itt bemutatott USB-TTL konverter CH340G chip alapú. Ez a célnak megfelelő, olcsó, megbízható típus. Csatlakoztassuk a konvertert a számítógép USB csatlakozójára, Windows 10 esetén a driver automatikusan letöltődik és települ, más esetben kézzel is telepíthetjük. Amikor az eszköz használatra kész, az eszközkezelő menüjében megtaláljuk, melyik virtuális soros port lesz az USB eszközhöz rendelve. Pl. COM10.



7.ábra

A szoftver feltöltése két fázisra bontható. Elsőként a ún. firmware-t kell feltölteni a modulra, amely nagyon leegyszerűsítve a magasszintű programozási környezetet fogja biztosítani számunkra. Az ESP8266-ra többféle ilyen firmware környezet is létezik, mi ebben a projektben a NodeMCU firmware-t használjuk. Ez biztosítja a legegyszerűbb programozási lehetőséget a modul számára, így a mozdonyt működtető szkript viszonylag rövid, jól áttekinthető, tovább fejleszthető, más feladatokra is mintaként felhasználható. Itt hívnám fel a figyelmet arra, hogy a projekthez felhasznált összes szoftver komponens, beleértve a forráskódokat is, letölthető a projekt GitHub oldaláról: https://github.com/kovtam/Locontrol, ahonnan további információk képek, videók, internetes weboldalak is elérhetőek a cikkel kapcsolatban. A NodeMCU firmware file feltöltéséhez először a fw-file-t kell letölteni a számítógépre. Ennek egy aktuális, felhasználásra kész, lefordított verziója megtalálható a fent említett projekt NodeMCU_FW könyvtárában, verziósszáma 0.9.6, e cikkhez kapcsolódóan ennek a használata javasolt, ill. tesztelt. A legfrissebb verzió letölthető a www.nodemcu.com oldalról is, megfelelő procedúra végrehajtása után. Ezen az oldalon a firmware teljes, igényes, példaprogramokkal is kiegészített online dokumentációja is elérhető. Szükség lesz még egy fw feltöltő segédprogramra **'ESP8266Flasher.exe'**, amelyet NodeMCU_Flasher könyvtárban találunk meg. A 7. ábrának megfelelő összeállítás után, a számítógépre csatlakoztatott USB-TTL konverterrel indítsuk el programot, válasszuk ki a megfelelő COM portot. Nyomjuk meg a nyomógombot, vagyis testeljük a GPIO 0 lábat. Ha minden stimmel, a program felismeri az ESP8266-ot és kiírja az adott IC-hez tartozó két, egyedi MAC címet és egy QR kódot. A 'Config' menüben az első sorba tallózzuk be a feltöltendő firmware file-t (***nodemcu_integer_0.9.6-dev_20150704.bin*** elnevezésű file). Lépjünk vissza az 'Operation' menübe és nyomjuk meg a Flash gombot. Várjuk meg amíg a feltöltés befejeződik, a gombot ezt követően engedjük fel. A 8. ábra képsorozata szemlélteti a folyamatot. Amennyiben nincs hibaüzenet a feltöltés sikeres volt, a programot bezárhatjuk és táp reset után a panel készen áll a felhasználói programok futtatására.

8.ábra

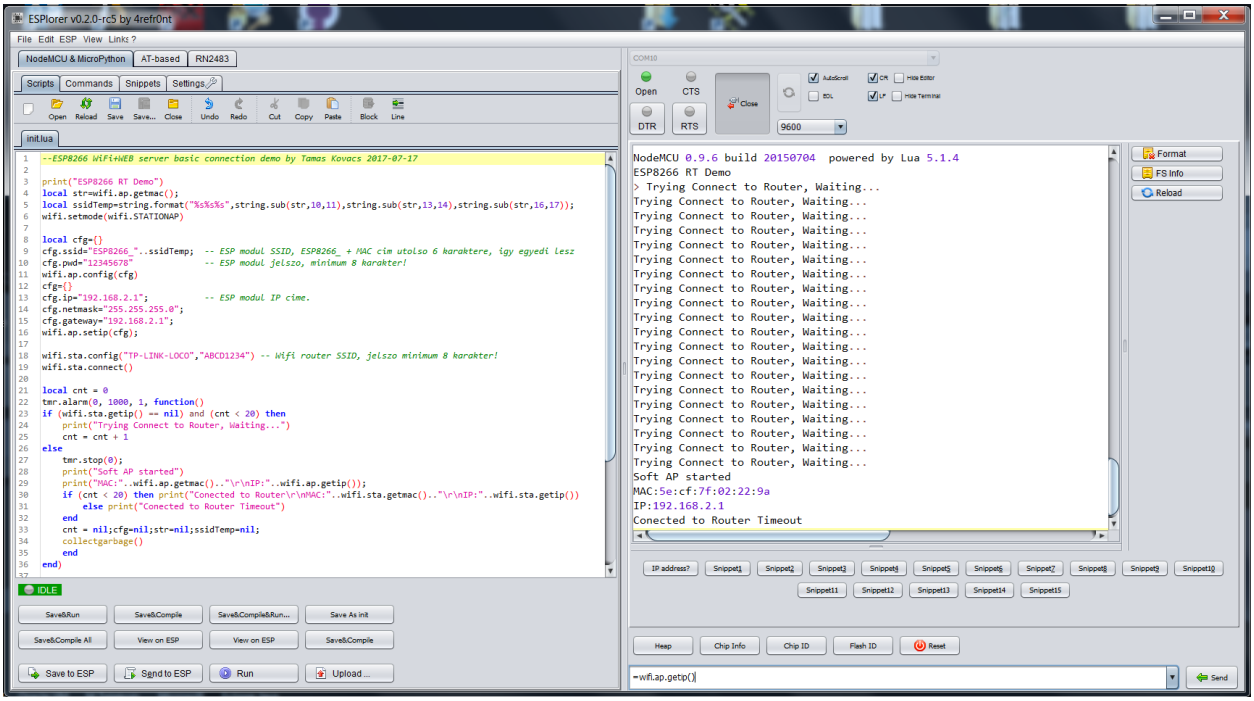
A következő fázis a működtető szkript file-ok feltöltése. Itt azonban nézzük át mit is tudnak ezek a szkriptek illetve milyen szoftver elemeket használunk fel a mozdony vezérléséhez. Mint a címből is kitűnik a vezérlés WiFi -n keresztül történik. A WiFi-t valójában mindenki ismeri, használja nap mint nap, de mégis érdemes összefoglalni mi is az valójában és miért alkalmazzuk ebben projektben, holott például egyszerű rádiótávirányítást is alkamazhattunk volna helyette stb. A Wifi, egy fejlett rádiós adatátviteli protokoll ún. fizikai rétege, amely internetes applikációkra van optimalizálva, így kiszolgálja a hozzá kapcsolódó magasabb szintű, felhasználói hálózati rétegeket, mint TCP/IP illetve az a felett elhelyezkedő HTTP protokollt használó alkalmazásokat. Az igazi előnye azonban abban mutatkozik meg, hogy ezeket a protokollokat a WiFi rádiós egységgel együtt minden mobil eszközben megtalálhatjuk. Az ESP8266 szintén implementálja mindezeket, ezért a kommunikáció a mobil eszköz és az ESP8266 modul között magas szinten, rendkívül egyszerűen megoldható, minden kiegészítő hardver elem nélkül is. Továbbá az általunk használt NodeMCU firmware tartalmaz egy komplett web szervert megvalósító szoftver blokkot is, amely segítségével, a mobil eszköz web böngészőjével, HTTP protokoll-on keresztül létesíthetünk két irányú adatkapcsolatot, célszerűen egy erre tervezett weboldal révén. Az adatok küldésére szerverre ill. fogadására a szervertől (a szerver esetünkben az ESP8266 modul) a HTTP protokoll GET parancsát fogjuk használni. A webböngészők szintén a GET parancsot alkalmazzák a weboldalak letöltésére a webszerverekről. A fentiek alapján most nézzünk egy példát, hogyan tudunk egy webszervert indítani az ESP8266 modulunkon, amely automatikusan felkapcsolódik egy WiFi router-re, és DHCP-n keresztül kap a router-től egy IP címet pl.: 192.168.0.1, majd szolgáltat egy egyszerű weboldalt, ami jelen estetben csak egy üdvözlő szöveg: **Hello, NodeMCU!** Ennek kipróbálásához a 9.ábrán látható lua szkriptet kell feltöltenünk a modulba és futtatni. A kód két fő részre bontható. Az első részben a WiFi, rádiós kapcsolatot hozzuk létre, a másodikban pedig a web szervert, amely már a weboldalunk tartalmát is magába foglalja, mindezt összesen 50 sor kóddal! A program kód elemzése

```lua
--ESP8266 WiFi+WEB server basic connection demo by Tamas Kovacs 2017-07-17

print("ESP8266 RT Demo")
local str=wifi.ap.getmac();
local ssidTemp=string.format("%s%s%s",string.sub(str,10,11),string.sub(str,13,14),string.sub(str,16,17));
wifi.setmode(wifi.STATIONAP)

local cfg={}
cfg.ssid="ESP8266_"..ssidTemp;  -- ESP modul SSID, ESP8266_ + MAC cim utolso 6 karaktere, igy egyedi lesz
cfg.pwd="12345678"              -- ESP modul jelszo, minimum 8 karakter!
wifi.ap.config(cfg)
cfg={}
cfg.ip="192.168.2.1";           -- ESP modul IP cime.
cfg.netmask="255.255.255.0";
cfg.gateway="192.168.2.1";
wifi.ap.setip(cfg);

wifi.sta.config("TP-LINK-LOCO","ABCD1234") -- Wifi router SSID, jelszo minimum 8 karakter!
wifi.sta.connect()

local cnt = 0
tmr.alarm(0, 1000, 1, function()
if (wifi.sta.getip() == nil) and (cnt < 20) then
    print("Trying Connect to Router, Waiting...")
    cnt = cnt + 1
else
    tmr.stop(0);
    print("Soft AP started")
    print("MAC:"..wifi.ap.getmac().."\r\nIP:"..wifi.ap.getip());
    if (cnt < 20) then print("Conected to Router\r\nMAC:"..wifi.sta.getmac().."\r\nIP:"..wifi.sta.getip())
        else print("Conected to Router Timeout")
    end
    cnt = nil;cfg=nil;str=nil;ssidTemp=nil;
    collectgarbage()
    end
end)

-- Web HTTP server part

srv=net.createServer(net.TCP)
srv:listen(80,function(conn)
    conn:on("receive", function(client,request)
        local buf = "";
        buf = buf.."HTTP/1.0 200 OK\r\nContent-Type: text/html\r\n\r\n<h3> Hello, NodeMCU!</h3>";
        client:send(buf);
        client:close();
        collectgarbage();
    end)
end)

```

9.ábra

röviden a következő. A lua ún. objektum orientált programnyelv. Ez nagyon leegyszerűsítve az jelenti, hogy benne, logikailag összetartozó elemeket, olyan halmazként definiálunk, amelyek tartalmaznak adatokat, és műveleteket is képesek végezni az adataikkal. Ezen felül kommunikációs lehetőséget és adat cserét is tudnak biztosítani másik ilyen halmazokkal. Ezeket a halmazokat nevezzük objektumoknak. Az előbbi példa kódjában a „wifi” objektum már létezik, amin egy művelet elvégzését kérjük, szaknyelven meghívjuk a wifi objektum egy metódusát, amivel beállítunk vele egy adatot, szaknyelven változót, ami a wifi objektumot most arra állítja be, hogy STATIONAP, azaz router közbeiktatott hálózati STATION, illetve un. direkt, Access Point AP, hibrid módban működjön. A hibrid mód itt azt jelenti, hogy mindkét mód egyszerre aktív, tehát az ESP modulhoz routeren keresztül, illetve router közbeiktatása nélkül, direkt módban is tudunk kapcsolódni a mobil eszközzel. Az objektum neve után ponttal elválasztva adjuk meg a metódus nevét, a zárójelben pedig a beállítani kívánt változó értéke van megadva: `wifi.setmode(wifi.STATIONAP)`. A (8-16) sorokban az AP, direkt módhoz tartozó beállításokat adjuk, meg egy cfg nevű adatstruktúra változó segítségével. A következő két sorban (18-19) annak a WiFi router-nek az SSID-ját, illetve jelszavát állítjuk be, amelyhez az ESP8266-os modullal csatlakozni szeretnénk. Úgy is mondhatjuk, a wifi objektum, az ESP8266-modul WiFi áramkörének

szoftveres leképezése, szakszóval reprezentációja. A wifi objektumból egy darab lehet, és azt a rendszer automatikusan létrehozza, mi csak a tulajdonságait, paramétereit állíthatjuk be metódusokon keresztül. A második részben (38-50) sorok, arra látunk példát, mikor egy objektumból mi hozunk létre egy új objektumot, amely révén a webszervert valósítjuk meg. A net objektum, hasonlóan a wifi objektumhoz automatikusan létrejön. A net objektum createServer nevű metódusával hozhatunk létre egy Server objektumot. Ilyen szervert, elvileg többet is létrehozhatunk, ezért ezt, a most létrehozni kívánt szervert egyedi névvel kell ellátni, neve ebben az esetben srv. Úgy mondjuk, hogy az srv, a Server objektum egy példánya. Létrehozásához szükséges kód: `srv = net.createServer(net.TCP)`. A zárójelben levő paraméter utal arra, hogy TCP/IP szerver-t szeretnénk létrehozni, a weboldal kiszolgáló részére. A további kódsorok a létrehozott, új szerver példány-t aktiválják, a srv:listen(… ún. csoportos paraméter beállítások alkalmazásával, aminek lényege több metódushívás és paraméter beállítás összevonása a kódban. Eredmény képen egy olyan TCP/IP szervert indítottunk a rendszerben, (ESP8266 modulban), amely a 80-as TCP-porton (alpértelmezett HTTP port) figyeli (szószerinti fordítás szerint: listen/hallgatja) a bejövő kéréseket, és ha van ilyen, akkor egy un. TCP/IP socket-et /sck/ (csatornát) épít fel a kliens felé és elküldi (client:send) neki a weboldal tartalmát HTTP protokoll szerint, amit a kliens, a mi esetünkben egy mobil eszköz, mobiltelefon vagy tablet, a webböngészőjében megjelenít a felhasználónak. Az srv server, indítása után a háttérben futó, ún. programszálként (thread) viselkedik, ami azt jelenti számunkra, hogy a programunk további részében számíthatunk annak szolgáltatásaira, amig az `srv.close()` paranccsal le nem állítjuk. Nézzük most meg, hogyan tölthetjük fel, és próbálhatjuk ki ennek a lua szkriptnek a működését a gyakorlatban. A lua programok feltöltéséhez, illetve a teszteléséhez az ESPlorer nevű ún. integrált fejlesztői környezetre (IDE Integrated Development Environment) lesz szükségünk. Ez egy Java-ban irt program, a zip file-t kicsomagolva, az *esplorer.jar* file-t futtatva elindul, külön installálni nem kell. Az IDE futási képe a 10. ábrán látható. Csatlakoztassuk a modul USB-TTL konverterrel a számítógéphez, Nyomógomb legyen kiengedett állapotban, GPIO0 láb H szinten.

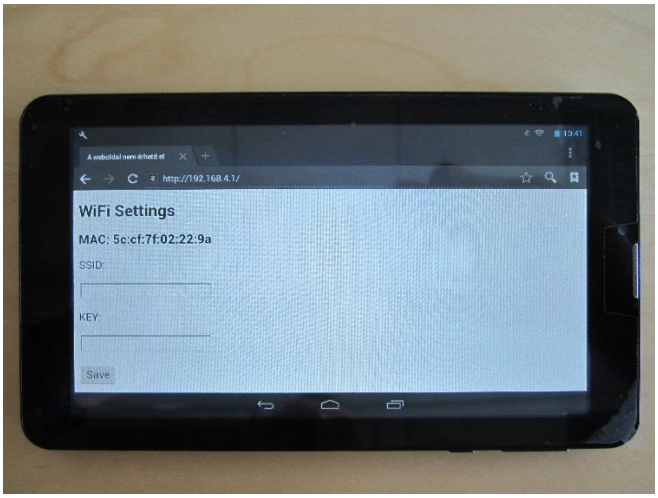


10.ábra

Az ESPlorer IDE bal oldali ablakában egy text editor található, amiben a lua szkripteket lehet megnyitni, editálni, a jobb oldali ablak gyakorlatilag egy soros terminál, ami az ESP8266-modullal való kommunikációra szolgál. A COM port és adatsebesség 9600 b/s beállítás után az OPEN gombra nyomva lépjünk kapcsolatba az ESP8266 modullal. A text editorban nyissuk meg az ***init.lua*** file-t, ami a 9. ábrán is látható lua kódot tartalmazza. Írjuk át a kódban (18.sor) a WiFi router-ünk SSID-jét ill. jelszavát és mentsük el. **Fontos!**: Az ESP modul alapértelmezetten WPA2 titkosítást használ a WiFi rádiós hálózaton, tehát a router-en is ezt kapcsoljuk be és minimum 8 karakter legyen a jelszó hossza! Ezután a SAVEtoESP gombbal töltsük fel az ESP modulra. Az ESP modul mindig az ***init.lua*** nevű file-t fogja először futtatni reset után. A IDE-n a Reset gomb megnyomása után, a szkript futni kezd, majd megpróbál csatlakozni a WiFi router-hez, amennyiben ez sikerül kiírja: Connected to Rouer és az IP címet, amit a router kiosztott számára. A WiFi router menüjében megtaláljuk az ESP modult, a DHCP clients menüpontban, itt is megtudhatjuk milyen IP címet osztott ki a router az ESP modul számára. pl: 192.168.100.1
A szintén a rouer-hez csatlakozó mobil eszközön nyissunk meg egy böngésző ablakot, majd ennek címsorába írjuk be az ESP modul IP címét. A böngésző megjeleníti az ESP modulban tárolt weboldalt! A modul IP címét az ESP modul terminálból is kiírathatjuk, a: `print('IP: ',wifi.sta.getip());` parancs segítségével. A szkript is minden futtatáskor ki fogja írni a terminálra az aktuális IP címet. (Wifi router hiányában, az ESP modulhoz, direkt módban is csatlakozhatunk, jelen esetben a 192.168.2.1 IP címmel.)

A mozdony teljes programkódja négy lua file-ból áll: ***loco.lua, setwifi.lua, wifi.lua, init.lua***. Ezeket nyissuk meg az IDE-ben, majd sorban töltsük fel ezeket az ESP modulra, ügyelve, hogy más file-ok ne legyenek rajta, ha mégis vannak, azokat először töröljük le. Futtassuk a programot, majd első lépésként direkt módban kapcsolódjunk a mozdonyhoz, a WiFi hálózat neve (SSID): LocoC, jelszó: 12345678. Az IP cím: 192.168.4.1, ezt írjuk be a böngésző címsorába. A böngészőben, egy konfigurációs weboldal jelenik meg. 11.ábra. Itt kell megadnunk a Wifi router SSID-jét és jelszavát, amit használni kívánunk a többi mozdonyaink ill. a terepasztal vezérléséhez. Az adatokat a program egy kétsoros szöveg file-ba menti, az ESP modulra, és ezen adatok felhasználásával a következő indulásnál már az itt beállított WiFi router-hez fog kapcsolódni a mozdony automatikusan. Természetesen ezen beállítások a fő oldalról is elérhetőek és szükség esetén megváltoztathatók. A már beállított adatokkal rendelkező mozdony, a Wifi router által kiosztott, IP címen, a 12.ábrán is látható weboldalról lesz irányítható. A weboldalon Start gombbal a mozdony elindítható, sebessége a -, + gombokkal növelhető, ill. csökkenthető. A REW gombbal a menetirány változtatható meg. Ilyen esetben a vonat lelassul, megáll majd az ellenkező irányba fokozatosan felgyorsul arra a sebességre, amin az előző irányban haladt. A HLT! gomb azonnali megállást vált ki. A weblapról az aktuális sebesség és irány is leolvasható. A Reset Wifi gombbal beléphetünk az előbb már ismertetett konfigurációs weboldalra. Láthatjuk, hogy amennyiben több ilyen WiFi-s mozdonyunk is van, a WiFi router mindegyiknek más IP címet fog kijelölni. A WiFi router-eknek van egy olyan szolgáltatása, hogy egy MAC address-hez mindig ugyanazt az IP címet rendelik hozzá. A MAC cím minden ESP modulnak egyedi, ezért érdemes ezt a szolgáltatást igénybe venni. Az adott mozdony MAC címét a konfigurációs weboldalról is leolvashatjuk. Így elérhetjük, hogy egy mozdony mindig ugyanazon az IP címen legyen elérhető. Ez kényelmesebb kezelést biztosít számunkra ugyan, de így is annyi böngészőablakot kell megnyitnunk, ahány mozdonyt kívánunk egyszerre irányítani, persze több mobilról is irányíthatjuk a mozdonyokat egyszerre. Itt mutatkozik meg a WiFi és az IP technológia előnye, mert ezt hagyományos rádiótávirányítás esetén csak többcsatornás eszközök összehangolt rendszerében tudnánk megvalósítani. A webes mozdony vezérlés motorja a ***loco.lua*** file-ban található kód. A kódot a 13.ábrán találhatjuk, sok ismerős elemet találunk benne pl. a web szerver létrehozását, weboldal felépítést stb. Egy új elemet érdemes kiemelni, ami a HTTP szerver része, ez pedig a HTTP protokoll GET parancsának implementálása. A weboldalon keresztül ezzel a paranccsal tudunk adatokat elküldeni a web szervernek ill. lekérni onnan. A programunk vezérlési algoritmusának kulcs eleme a GET parancs. A

HTTP gyakorlatilag szöveg alapú protokoll. Amikor a weboldalon egy vezérlő gombot megnyomunk, a böngésző egy karakter sort, szöveget, küld a webszerver, a mi esetünkben az ESP modult tartalmazó mozdony felé. A szerver oldalon a html szövegből ún. parse-olással (szövegmintára való szűréssel) megkeressük a weboldalon lenyomott gombhoz tartozó transfer változó(pin) értékét (6-17 sorok), majd a végrehajtást "if" elágazás feltételekkel átadjuk a megfelelő kezelő funkciók számára (24-43 sorok).



11.ábra 12.ábra

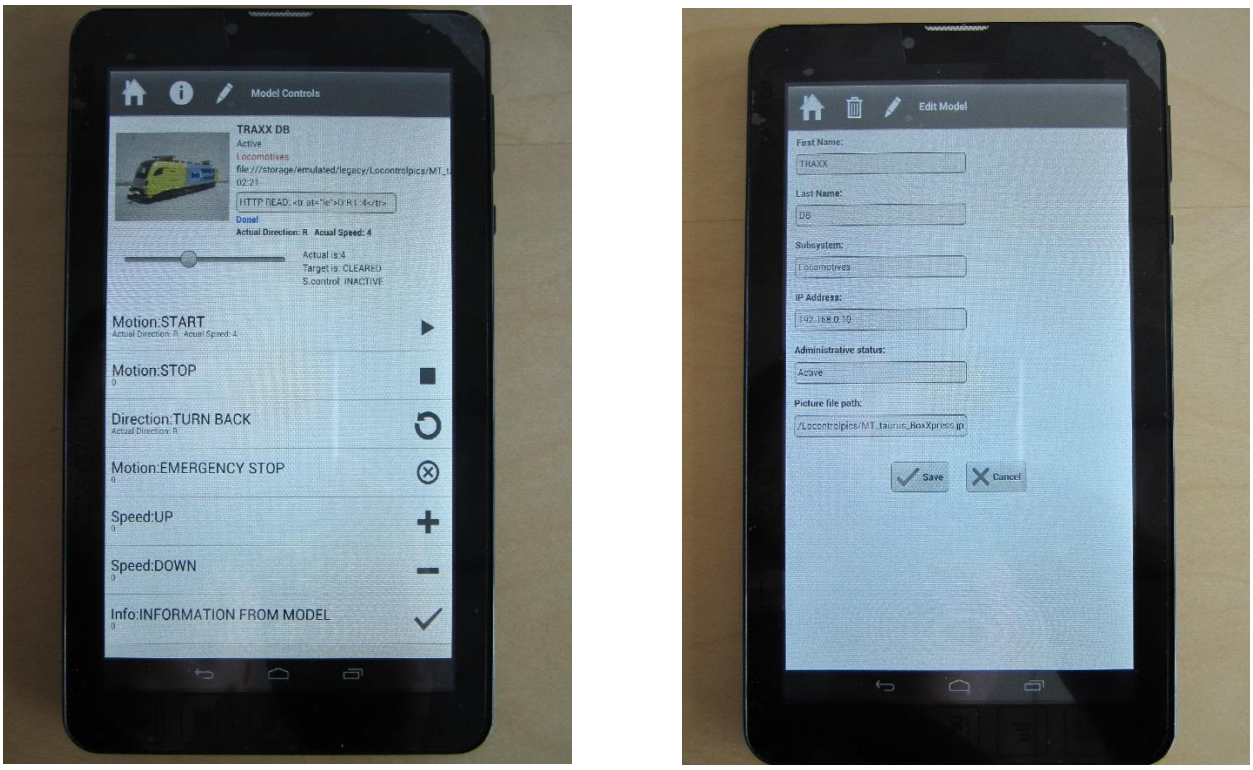
Példaként nézzük meg egy vezérlési folyamat algoritmusát, a többi is teljesen hasonlóan működik. A START gomb megnyomásakor a vezérlési változó pin="START" lesz. Ilyenkor a szerveroldalon a 24. sorban levő szűrőfeltétel, "igaz" esetben, a `start_handler()` nevű függvényt hívja meg (78.sor). A start_handler funkció működése: A "level" nevű globális változó a mozdony sebesség adatát tárolja, az egész START folyamat csak akkor végrehajtható, ha ez a változó 0 értékű, vagyis a mozdony áll (80.sor). A "direction" nevű szintén globális változó a mozdony irány adatát tárolja 0 értékű előremenet, 1 értékű hátramenet esetén. Az aktuális menetirány vizsgálata alapján (84.sor), ha 0 az érték beállítjuk az előremenethez tartozó PWM objektum paramétereket (87-90 sorok). Majd vezéreljük a menetfény LED-eket a menetiránynak megfelelően (92-93 sorok). Amennyiben a menetirány ellentétes, a (100-106 sorokban) található hasonló kód fut majd le. Végül a `deltaspeed_handler_up()` (110.sor) funkció kerül meghívásra, ami a PWM jel kitöltési tényezőjét folyamatosan változtatva (156-167 sorok) emeli a kívánt fordulatszámra a meghajtómotor fordulatszámát, ezzel gyorsítva fel a mozdonyt a megfelelő sebességre.

Az alkamazás során szükségünk van arra, hogy a mozdonyból az aktuális sebesség ill. irány adatokat le tudjuk kérdezni. Például akkor, ha böngészőt bezárjuk és az adatkapcsolat megszakad, majd új böngészőt nyitunk meg és újra kapcsolódunk az egyik mozdonyhoz. Ezért a weboldalba injektáljuk a következő ún. xml struktúrát (61. sor): `<tr at="le">D:"..directionp.." L:"..levelp.."</tr>`
Ennek az adat beágyazásnak az az előnye, hogy a weboldalon minden egyéb formázás nélkül kiírja a benne foglalt adatokat, jelen esetben pl. **D:F L:3** 12.ábra. Ez esetben az xml adatstruktúra így néz ki:
`<tr at="le">D:F L:3</tr>`
A mobilalkalmazásunkban pedig szintén előnyös lesz, mert az ilyen, xml formátumú adatstruktúra rendkívül egyszerűen feldolgozható, ezért ebben az esetben is hatékonyan tudjuk alkalmazni web alapú adatlekérések esetében.

```
-- LocoDecoder by Tamas Kovacs Date:2017-07-20 15_45 Ver:1.2

srv=net.createServer(net.TCP)
srv:listen(80,function(conn)
    conn:on("receive", function(client,request)
        local buf = "";
        local head = "HTTP/1.1 200 OK\n\n<!DOCTYPE HTML>\n<html>\n<head></head>\n<body>\n";
        local _, _, method, path, vars = string.find(request, "([A-Z]+) (.+)?(.+) HTTP");
        if(method == nil)then
            _, _, method, path = string.find(request, "([A-Z]+) (.+) HTTP");
        end
        local _GET = {}
        if (vars ~= nil)then
            for k, v in string.gmatch(vars, "(%w+)=(%w+)&*") do
                _GET[k] = v
            end
        end
        buf = head;
        buf = buf.."<h1> LOCOntrol";
        buf = buf.."<p>Motion <a href=\"?pin=START\"><button>START</button></a> <a href=\"?pin=STOP\"><button>STOP</button></a></p>";
        buf = buf.."<p>Direction <a href=\"?pin=REW\"><button>REW</button></a> <a href=\"?pin=HLT\"><button>HALT!</button></a></p>";
        buf = buf.."<p>Speed <a href=\"?pin=SLOW\"><button>-</button></a> <a href=\"?pin=FAST\"><button>+</button></a></p>"

        if(_GET.pin == "START")then
              start_handler();

        elseif(_GET.pin == "STOP")then
              stop_handler();

        elseif(_GET.pin == "REW")then

            if (direction == 0) then
                dreverse ();
            elseif (direction == 1) then
                dforward ();
            end

        elseif(_GET.pin == "HLT")then
              halt ();
        elseif(_GET.pin == "SLOW")then
              slow_maker ();
        elseif(_GET.pin == "FAST")then
              fast_maker ();

        end

        local levelp = 0;
        local directionp = "";

        if (level > 0) then
            levelp = level/100

        end

        if (direction == 0) then
              directionp = "F"
        elseif (direction == 1) then
             directionp = "R"
        end

        buf = buf.."<tr at=\"le\">D:"..directionp.." L:"..levelp.."</tr>";

        buf = buf.."<form src=\"/\"><br><button type=\"submit\" name=\"set\" value=\"RST\">Reset WiFi</button></form>\n";
        buf=buf.."</body></html>";
        client:send(buf);
        client:close();
        collectgarbage();

        if(_GET.set == "RST")then
           file.remove("wificonf");
           node.restart();
        end
  end)
end)

-- functions ---------------

function start_handler ()

   if (level == 0)then

        level=200
```

```
        if (direction == 0) then


            pwm.stop (ib)
            delay_ms (1000)
            pwm.start (ia)
            direction=0

            gpio.write (led2, gpio.LOW);
            gpio.write (led1, gpio.HIGH);



         elseif (direction == 1) then


            pwm.stop (ia)
            delay_ms (1000)
            pwm.start (ib)
            direction=1

            gpio.write (led1, gpio.LOW);
            gpio.write (led2, gpio.HIGH);

        end

        deltaspeed_handler_up ()

    end

end

function stop_handler ()

    level=0
    deltaspeed_handler_down ()
    gpio.write (led2, gpio.LOW);
    gpio.write (led1, gpio.LOW);

end

function fast_maker ()

    if (level<800) then
    level=level+100
    deltaspeed_handler_up ()
    end

end

function slow_maker ()

    if (level>=100) then
    level=level-100
    deltaspeed_handler_down ()
    end

end

function deltaspeed_handler_up ()

   local dutyval = 0
   local speednow = level
   local directionl = ia

   if (direction == 1)
   then directionl=ib;
   end

   dutyval = pwm.getduty(directionl)


    if (speednow > dutyval) then

      while (speednow > dutyval) do

        dutyval = dutyval + 100

        pwm.setduty(directionl, dutyval)

        delay_ms (500)

        end
    end
end

function deltaspeed_handler_down ()

   local dutyval = 0
   local speednow = level
   local directionl = ia

   if (direction == 1) then
   directionl=ib;
   end

   dutyval = pwm.getduty(directionl)


   if (speednow < dutyval) then

      while (speednow < dutyval) do

        dutyval = dutyval - 100

        pwm.setduty(directionl, dutyval)

        delay_ms (500)

        end
    end
end

function dforward ()

    if (direction == 1)then

    local levelmemo = level

    stop_handler ()

    pwm.stop (ib)
    delay_ms (1000)
    pwm.start (ia)
    direction=0

    gpio.write (led2, gpio.LOW);
    gpio.write (led1, gpio.HIGH);

    level=levelmemo
    deltaspeed_handler_up ()

    end
end

function dreverse ()

    if (direction == 0)then

    local levelmemo = level

    stop_handler ()

    pwm.stop (ia)
    delay_ms (1000)
    pwm.start (ib)
    direction=1

    gpio.write (led1, gpio.LOW);
    gpio.write (led2, gpio.HIGH);

    level=levelmemo
    deltaspeed_handler_up ()

    end
end

function halt ()

    level=0
    pwm.setduty(ia, 0)
    pwm.setduty(ib, 0)

end

function delay_ms (milli_secs)
   local ms = milli_secs * 1000
   local timestart = tmr.now ()

   while (tmr.now () - timestart < ms) do
      tmr.wdclr ()
   end
end
```

13.ábra



14.ábra

A mobilapplikáció az Adobe AIR elnevezésü fejlesztői környezetben lett kivitelezve. Ennek egyik nagy előnye, hogy itt fejlesztett alkalmazás fut Google Android, Apple iOS, ill WebOS operációs rendszerű mobileszközökön egyaránt. További előnye, hogy a programozási technikája lényegesen egyszerűbb, mint mondjuk a natív Android fejlesztőkörnyezetben használt Java alapú kódolás. Szerencsére magyar nyelvű szakirodalom is rendelkezésre áll, *Fehér Krisztián: Androidos szoftverfejlesztés alapfokon*. Ez a könyv részletesen foglalkozik az Adobe AIR mobil programozási témával. A mobilapphoz tartozó forráskód és egyéb hasznos infók azzal kapcsolatban a LOCONTROL_MOBILEAPP nevű könyvtárban találhatóak. Az Adobe AIR hivatalos fejlesztőkörnyezete az Adobe FlashBuilder nevű IDE, ez azonban fizetős szoftver, ugyan egyhónapos próbaverziót le lehet tölteni, mégis az ingyenesen is használható, majdnem egyenértékü FlashDevelop nevű IDE-t javaslom azoknak, akik szeretnének efajta mobilapp fejlesztéssel foglalkozni. A mobilapp Androidra lefordított, installálásra kész változata a LOCONTROL_ANDROID könyvtárban található. A ***LOCOnew001.apk*** nevű file-t másoljuk a mobileszköz-re, file manager segítségével, majd telepítsük. A telepítésnél az Android figyelmeztet, hogy az Adobe AIR környezet is szükséges a programunk futtatásához és azt is telepítenie kell, ezt engedélyezzük neki. Az mobilapplikáció “LOCONTROL”, SQLite adatbázisban tárolja a mozdonyokhoz tartozó adatokat, pl. az IP címet is. Ezeket a beállításokat a mobilapp adatbázisában menti, és a rendszer újraindításakor autómatikusan betölti. Az első indítás alkalmával hozzunk létre egy adatbázist. A kezdőképernyőn nyomjuk meg a „+” ikont, majd a következő képernyőn a “ceruza” ikont, majd ezt követően megjelenő képernyőn a CREATE DB gombot, majd az OK-t. Az alkalmazásból lépjünk ki, majd indítsuk újra. A mozdonyokhoz képeket is hozzárendelhetünk. A kép file-okat, file managerrel a belső tárolón létrehozott Locontrolpics könyvtárba mentsük, majd a mozdony adatlapján “Edit model” képernyőn a

Picture file path: hoz adjuk meg az útvonalat: ***/Locontrolpics/mozdonykep1.jpg*** például, majd Save gombbal mentsük el. A modell sebességét a mobilapplikációból nem csak a -/+ gomb nyomogatásával, hanem egy tolópotméterhez hasonló csúszkával is szabályozhatjuk. A vezérlő menüből a pipa ikon megnyomásával lekérdezhetjük az adott mozdony aktuális sebességét és haladási irányát, ezzel aktualizálva a kijelzett értékeket a képernyőn. Kérem a kedves olvasót, hogy az itt közreadott alkalmazást ne tekintse professzionális, kereskedelmi célú projektnek. A cél elsősorban a téma iránt való érdeklődés felkeltése ill. ismeretterjesztés volt. A kidolgozásnál és tesztelésnél ügyeltem arra hogy minden működőképes legyen, azonban bőven lenne mit még fejleszteni, tökéletesíteni rajta. Ezzel kapcsolatban minden érdeklődőt bíztatnék a továbbfejlesztésre, illetve más projektek indítására a témával kapcsolatban. Felmerülő kérdéseket, visszajelzéseket szívesen fogadok. E-mail: *kovtam112@gmail.com*