Budapesti Műszaki és Gazdaságtudományi Egyetem
Mesterséges intelligencia (BMEVIMIA313)

Felhő alapú elosztott vezérlés beágyazott robot rendszeren

Házi feladat

Kovács Levente Ákos (CM6UKU),
Tóth Krisztián Dávid (J38GIK)
2015. dec. 11.

1 Bevezetés

1.1 A házi feladat célja

Egy labirintusban közlekedő reflexszerű (magától csak jobbra és felfele mozgó) robot szimulálása beágyazott rendszeren (Raspberry Pi-on), ami adott kezdeti állapotból egy vég állapotba próbál eljutni, egy PC-n futó cloud szerver segítségével. Továbbá az ehhez tartozó technológiák megismerése és ismertetése (Node-RED, Yakindu).

1.2 Program futása

A Robot a pályát, a vég céljának és a saját pozíciójának koordinátáit, a cloud rendszertől kapja meg az inicializációs szakaszban. Ez után a robot addig megy felfele ameddig csak tud, és ha elakad, akkor jobbra kerül. Amennyiben se jobbra, se fel nem tud lépni a robot (szenzorjai falat érzékelnek mind 2 irányban), akkor jelez a cloud rendszernek, hogy szüksége van segítségre és elküldi az aktuális koordinátáit, ezek után pedig vár a külső vezérlés válaszára. A cloud rendszer lefuttat egy A* útkereső algoritmust a robot és a cél koordinátáival. Majd vissza küldi a robotnak. A gyakorlati megvalósításokról külön fejezetekben részletesebben írunk.

1.3 Felhasznált Technológiák

1.3.1 A cloud rendszerhez

A kommunikációs logika Node-RED-ben került implementálásra MQTT (Message Queueing Telemetry Transport) protokoll felhasználásával, MQTT broker-nek pedig a mosquitto open-source alkalmazást használja. A kereső motornak IMOR nevű felhasználó pathfinding Node.js package-ét használja.

1.3.2 A robothoz

Yakindu-ban tervezett állapotgépből generált C++ kódot futtat, továbbá szintén Node-RED fut a kommunikáláshoz.

2 Technológiák

2.1 Node-RED

2.1.1 Bevezetés

A Node-RED egy grafikus felület hardware-ek, api-k, és online szolgáltatások összekötésére, és az eseményvezérelt kommunikációs modell létrehozására. A Node-RED alapja Node.js, ami megkönnyíti a fejlesztést a több mint 120 000 szabad forrású moduljával, ami gyorsan elérhető a npm-en keresztül (egy parancs cmd-ből), továbbá optimálissá teszi cloud-on és Raspberry Pi-on való felhasználásra.

A flow („esemény”) folyam): az összeköttetést és a hozzá tartozó logikát megvalósító eseményvezérelt modell. A Node-RED-ben ilyen flow-kat alakítunk ki, innentől kezdve a futó modellünkre/programunkra is flow ként fogok utalni.

Egy flow kialakítását web browser-ben (Firefox, Google Chrome, …) a Node-RED portjára (saját-ip:1880) csatlakozással lehet végrehajtani miután fut a Node-RED. Minden flow-t automatikusan ment a Node-RED, de csak ha deploy-oltuk. A flow-k között a képernyő tetején lévő sheet-ek (munkalapok) kiválasztásával tudunk váltogatni, és itt tudunk ujjat sheet-et létrehozni a + gombbal.

A flow–kat JSON-ben tárolja a Node-RED megkönnyítve az importálás/exportálás–ukat, továbbá a megosztásukat az „online flow library”-ban.

2.1.2 Telepítése

(Megj.: Jól érthető angol leírás található a http://nodered.org-on. Ennek az alfejezetnek csak a windows-os telepítésnek általános ismertetése a célja.)

Le kell tölteni és telepíteni egy 0.10.x-nél későbbi Node.js-t.

Majd a legegyszerűbb módszer, globálisan telepíteni a Node-RED-et, a node Package manager-en (npm-en) keresztül az alábbi paranccsal:

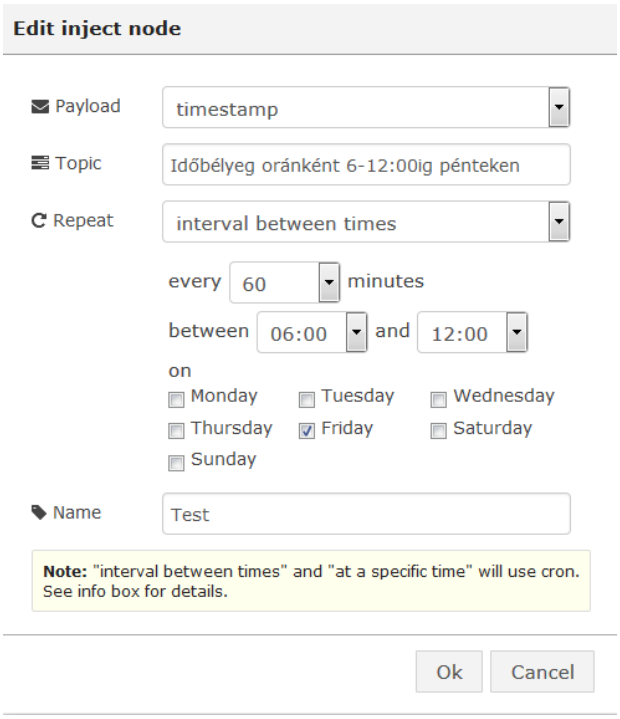
npm install -g --unsafe-perm node-red

Ez a parancs a C:\Users\”user-name”\AppData\Roaming\npm mappába fogja globálisan telepíteni a Node-RED-et, amit ez után bárhonnan futtatni lehet parancssorból a node-red parancs kiadásával.

2.1.3 Node-ok

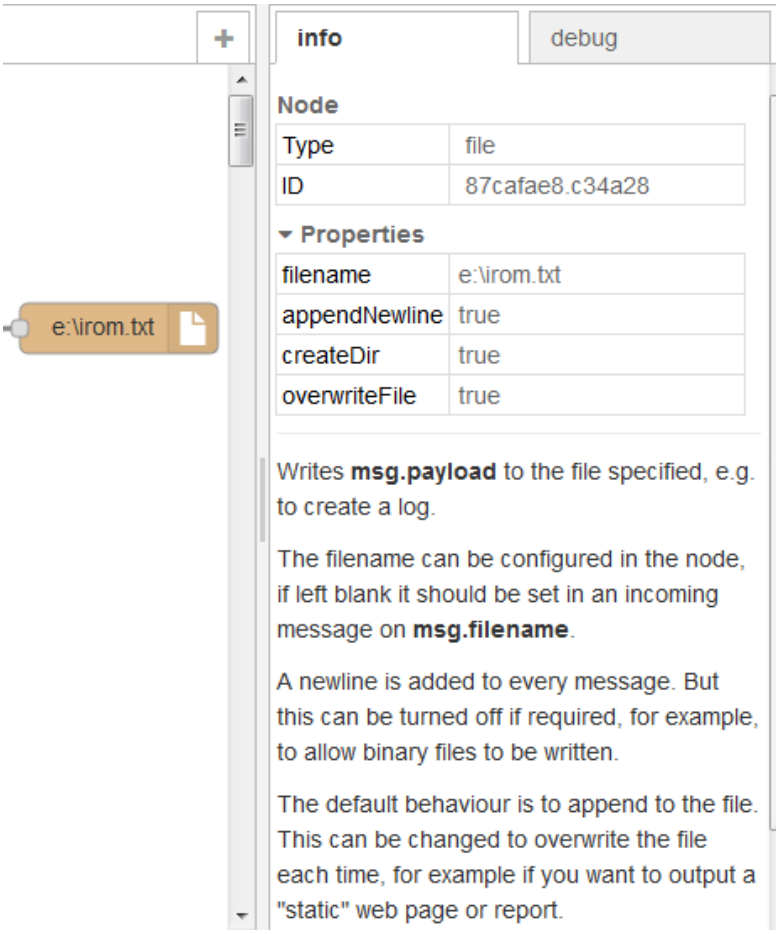
A node-ok, a folyamjainknak (flow) az építő elemei. Ezekből huzalozzuk össze a grafikus felületen a megvalósítandó rendszerünket. A node-ok működését úgy tudjuk beállítani, hogy kétszer klikkelünk a modul ikonjára és a felugró ablakon beállítjuk a megfelelő konfigurációs értékeket.

PL: Inject node beálltása



1. ábra

Minden node-hoz tartozik egy leírás, ezt a grafikus felület jobb oldalán található "info" ablakban olvashatjuk el, ez tartalmazza a node típusát, a program által generált egyedi azonosítóját, a properties listát, ami a node konfigurációs értékeit tartalmazza, és egy általánost leírást a node működéséről.

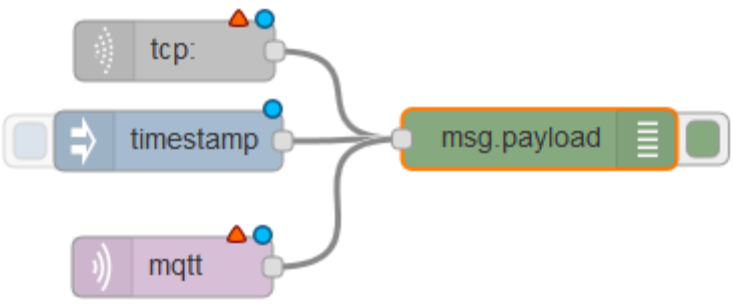


2. ábra File író node info palettája

2.1.4 Rendszerezésük

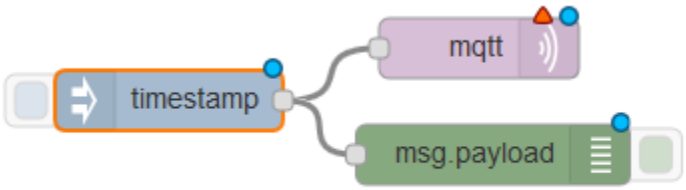
2.1.4.1 Funkció szerint

- **Input (Bemeneti):** Esemény (msg) generálásra használatos, ez lehet különböző protokollokról (Tcp/Udp/Mqtt) beérkező üzenetből létrehozott msg vagy lehetőség van msg injektálásra. Az esemény injektáló node-ot be lehet úgy állítani, hogy ismételje az üzenet injektálást bizonyos időközönként (pl. másodpercenként), megadott időpillanatban (pl. péntek 18:00-kor), vagy megadott intervallumon belül időközönként (pl. szombaton és pénteken 19:00-20:00 között percenként). Ez széleskörű polling lehetőségeket biztosít a felhasználó számára. Az injektáló node, default-ból időbélyeget generál, de konstans string-eket is lehet vele küldeni



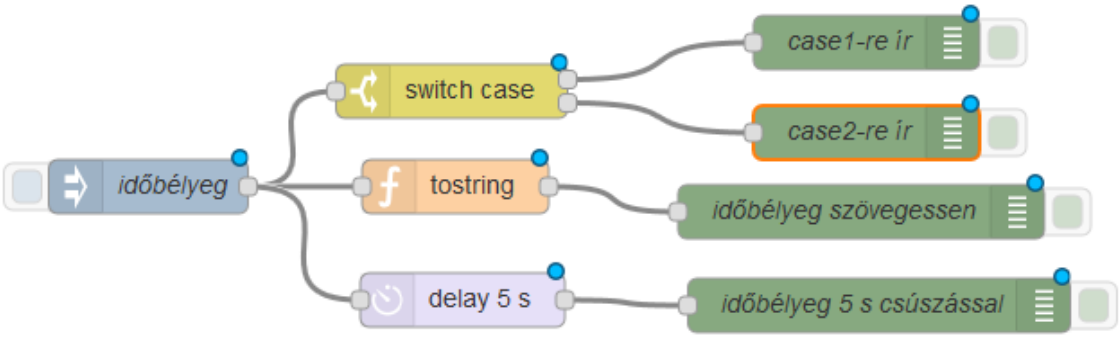
3. ábra (Bejövő üzenetek payload-ját kiírja a debug ablakra)

- **Output (Kimeneti):** Nyelő alfolyamat, él csak rá mutat belőle ki nem. Főleg üzenettovábbításra használjuk, pl. UDP, MQTT csomag küldése vagy debug node-ot (zöld) a message payload-jának logolásra.



4. ábra Időbélyeget küld ki MQTT protokollon keresztül, továbbá kiírja a debug ablakra

- **Function (függvény):** Olyan node-ok amik valamilyen adatmanipulációt (XML, Json convert to/from …), késleltetést (delay), vagy valamilyen kapcsolat felvételt (TCP/HTTP request) hajtanak végre. Ezeknek a node-oknak tipikusan kimeneten és bemenete is van. Itt található a saját kódot futató function node is, de ez saját sandbox környezetbe fut, aminek eredményeként nem hivatkozhat más package-re csak a saját magunk által Java scriptben megírt kód futhat benne. Ez főleg egyszerű, de egyedi adatmanipulációra jó, de a komplexebb feladatokhoz érdemes saját node-ot készíteni (lsd. később).

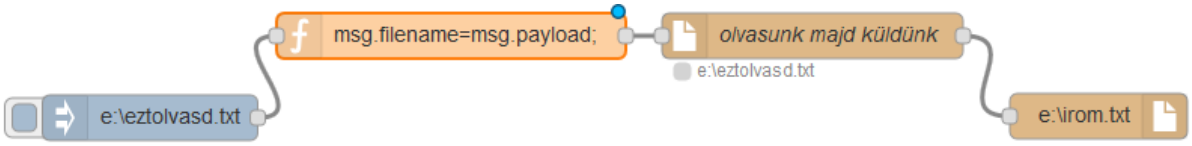


5. ábra Function node-ok használata.

- **Social:** Twitter/Email üzenetek lekérésére és küldésére használt node-ok.

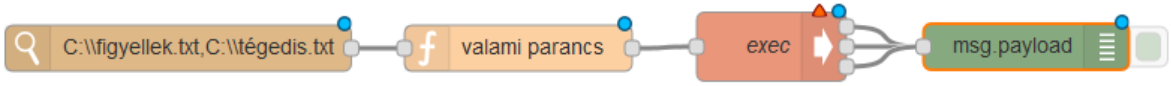
6. ábra Tweet-ekből e-mail-t küld, e-mail-eket tweet-eli

- **Storage:** File kezelést megvalósító 2 node tartozik alá, az 1. beolvassa azt a filet, aminek a nevét kapta msg payload-ban a bemenetén majd tovább küldi a tartalmát. A másik csak file-ba írást végez.



7. ábra Storage node-ok használata

- **Advanced:** Ide kerülnek a különleges funkciókat ellátó node-ok, továbbá általában az internetről letöltött node-ok. Gyárilag a rendszerhívást megvalósító exec (3 output stderr stdout return), és a fájl változást figyelő watch található itt.



8. ábra Advanced node-ok használata

2.1.4.2 Kimenetek/bemenetek száma szerint

Egy node-nak lehet tetszőleges számú kimeneti és 0 vagy 1 bemeneti socket-je (ezek kezeléséről lsd. később). Egy bemenet socket-be tetszőleges számú event folyhat be (pl.: Advanced példánál exec-ből mind a három kimeneti socket a debug 1 bemenetével van összekapcsolva), és egy kimeneti socket-ből akár több irányba is továbbíthatunk egyszerre msg-t (lsd. Output példánál). Több kimeneti socket-tel, külön választhatjuk az adatfolyamunkat több párhuzamos végrehajtási irányban, vagy feltételhez köthetjük az üzenet útját/tartalmát.

2.1.5 Node-ok készítése/felépítése

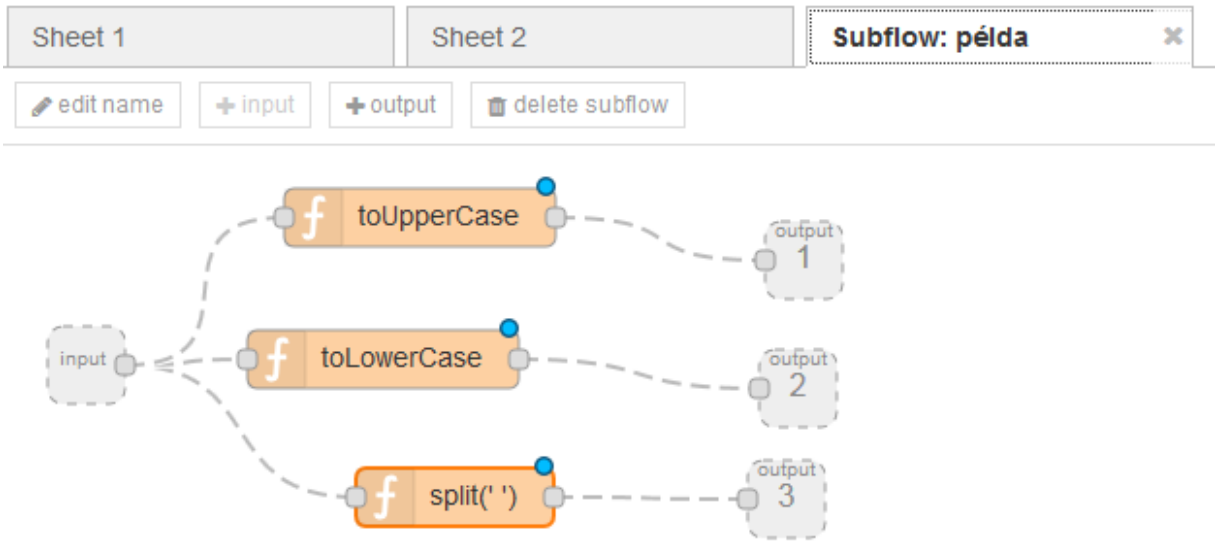
Egy node-ot három féleképpen lehet létrehozni.

2.1.5.1 Internetről npm-en keresztül

Ez a legegyszerűbb módszer, keresünk az npmjs.com oldalon található node-ok közül egy olyat, amire nekünk szükségünk van, majd kiadjuk a `npm install node-red-node-{filename}` parancsot és telepítjük (Megj.: minden node red node neve node-red-node-* al kezdődik, a többi sima node.js package ezeket a harmadik módszerben ismertetett módon lehet felhasználni.)

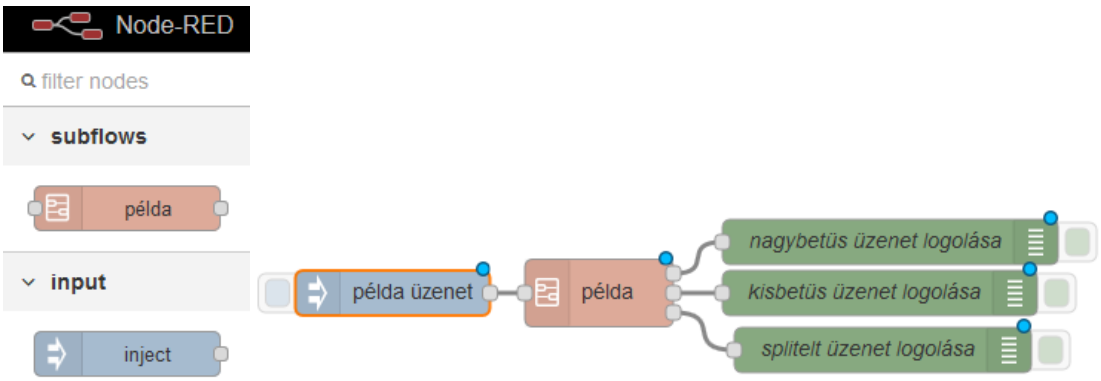
2.1.5.2 Subflow-ként

Lehetőségünk van a meglévő node-jainkból egy új node létrehozására, ez főleg az átláthatóságot és a hierarchikus tervezést hivatott megkönnyíteni. A Node-RED web browser-es kezelő felületén a jobb felső sarokban található 3 egymás alatti vízszintes vonalra klikkelve (), kiválasztjuk a create subflow opciót. Ezek után hozzá adhatunk maximum 1 bemenetet, és tetszőleges számú kimenetet a subflow-nk hoz, továbbá kialakíthatunk egy flow-t a hozzátartozó logikával.



9. ábra Subflow belseje

Ha ezzel készen vagyunk meg fog jelenni a node listában a subflow menü pont alatt az új node-unk. Innentől pedig használhatjuk, csak be kell húzni egy flow-ba.



10. ábra Subflow felhasználása

2.1.5.3 Manuálisan (Saját node készítése/node-ok felépítése):

Minden node két fájlból felépíthető manuálisan is. Egy HTML és egy JavaScript fileból. Ezek nevének meg kell egyeznie, és az alábbi formátumra illeszkedni: az első karakterek '–'-ig egy számkódot reprezentálnak, ezt követi a név.html/js attól függően, hogy melyik file (pl. 99-pelda.js , 99-pelda.html).

Ezeket a fileokat a C:\Users\"felhasználó név"\AppData\Roaming\npm\node_modules\node-red\nodes mappába kell elhelyezni, ha globálisan lett a Node-RED telepítve, ellenkező esetben a telepítési mappába.

A Node-RED működését és elérési útvonalait személyre lehet szabni (kellő hozzáértéssel), a settings.js file manipulálásával (ami a node red könyvtárban található).(Megj.: A node-ok készítéséről jó leírás és példa található a Node-RED weboldalán)

A HTML file: Ez a node konfigurációjáért felel, innen fogja tudni a browser-es grafikus felületünk, hogy hogyan kell kezelnie a node-ot, mi a bemenetek illetve kimenetek száma, milyen konfigurációs ablakot dobjon fel kiválasztáskor, melyik típusba sorolódik a node listában, hogy néz ki az ikonja, milyen leírást tartalmaz az info ablaka stb.

Példa HTML részlet:

```
<script type="text/javascript">
    RED.nodes.registerType('Pathfinding',{
        category: 'advanced',      // the palette category
        defaults: {             // defines the editable properties of the node
            name: {value:"Pathfinder mk1"},   //  along with default values.
            topic: {value:"nem fontos", required:false}
        },
        inputs:1,               // set the number of inputs - only 0 or 1
        outputs:3,              // set the number of outputs - 0 to n
        // set the icon (held in icons dir below where you save the node)
        icon: "function.png",     // saved in  icons/myicon.png
             color:"gray",
        label: function() {     // sets the default label contents
            return this.name||this.topic||"Pathfinding";
        },
        labelStyle: function() { // sets the class to apply to the label
            return this.name?"node_label_italic":"";
        }
    });
</script>
```

2.1.5.3.1 A JavaScript file:

A node viselkedését egy javascript file-ban tudjuk megírni. A function node-hoz képest meg van az az előnye hogy itt include-olhatunk(javascriptben require) más library-ket . (Megj.: Nagy könnyítést jelent, hogy rengeteg problémára található library az npmjs.com-on amit egy npm install paranccsal már használatra késszé is tehetünk.)

Alapvető js program keret:

```
module.exports = function(RED) {
    "use strict";
    // require any external libraries we may need....

       var PF = require('pathfinding');
    // The main node definition - most things happen in here
    function PathfindingNode(n) {
        // Create a RED node
        RED.nodes.createNode(this,n);
        this.topic = n.topic;
        var node = this;
        this.on('input', function (msg) {
```

Valamilyen logika, amit csinál a node, ez hívódik meg mikor msg érkezik a bemenetre.

```
        this.on("close", function() {
            // Called when the node is shutdown - eg on redeploy.
            // Allows ports to be closed, connections dropped etc.
            // eg: node.client.disconnect();
        });
    }

    // Register the node by name. This must be called before overriding any of the
    // Node functions.
    RED.nodes.registerType("Pathfinding",PathfindingNode);
```

```
}
```

2.1.6 Message kezelés

Node-RED-ben message-ekkel kommunikálnak a node-ok, ezek JSON objektumok amikre msg-ként hivatkozhatunk. Minden msg-nek van egy msg.payload property-je, ebben tároljuk a hasznos adatokat, ezen felül még msg.topic property-vel is rendelkezik sok default node által generált msg, ennek az oka a Node RED MQTT-s származásban keresendő. Vannak olyan node-ok amik több property-t is használnak, ilyen például a file („in”) node ami ha nem manuálissan kapja meg az olvasni kívánt file elérési útvonalát, akkor a msg.filename property-től várja ezt. De a saját node-jainkban is tetszőleges számú property-t bevezethetünk.(Megj.: a msg-k két node közötti property manipulációjára nagyon hasznos a function node, így pl. az msg.filename=msg.payload kód beiktatásával a file in számára feldolgozhatóvá lehet tenni egy sima msg-t)

2.1.6.1 Példa kódok üzenet manipulációra

Üzenet továbbküldése:

```
return msg;
```

Új üzenet létrehozása és továbbküldése:

```
var newMsg = { payload: "titkos üzenet" };
return newMsg;
```

Üzenet szétválasztása (Két output közül csak az egyiken továbbítja az üzenetet a másikon nem (nem küld NULL üzenetet, amelyik outputra null megy, azon megszakítja a folyamot))

```
if (msg.topic == "feltetel") {
   return [ null, msg ];
} else {
   return [ msg, null ];
}
```

Több üzenet egyszerre kiküldése/üzenet sorosítás: megvalósítható hogy egy adott outputra sorban több msg-t is küldjünk ki az alábbi példán szemléltetve:

```
var msg1 = { payload:"elso kimenete az output 1-nek" };
var msg2 = { payload:"masodik kimenete az output 1-nek" };
var msg3 = { payload:"harmadik kimenete az output 1-nek" };
var msg4 = { payload:"egyetlen uzenete az output 2-nek" };
return [ [ msg1, msg2, msg3 ], msg4 ];
```

Aszinkron üzenetküldés: ez nem szakítja meg a node futását csak kiküld egy üzenetet, de ilyenkor vigyázni kell, hogy a close event handler rendet rakjon utánunk.

```
var msg1 = {payload:"hello aszinkron vilag"};
node.send(msg1);
```

2.1.7 MQTT És a Node RED

Részletes angol nyelvű leírás található az alábbi linken, ennek a fejezetnek nem célja a részletes ismertetés, erre van a link:

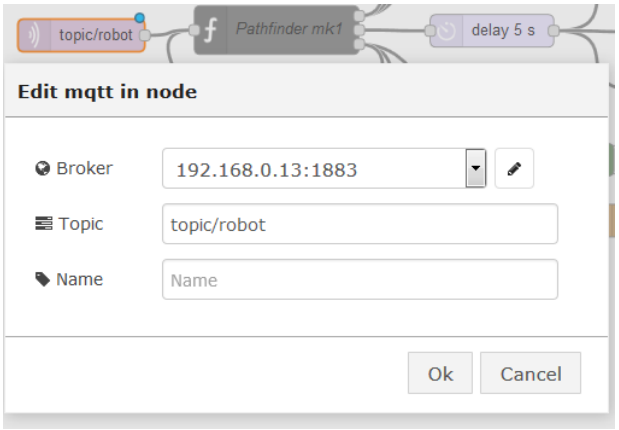
http://www.rs-online.com/designspark/electronics/eng/blog/building-distributed-node-red-applications-with-mqtt

Az MQTT egy általános topic based hálózati kommunikációs protokoll, amin keresztül eseményekre iratkozhatunk fel, vagy bizonyos témákban (topic-ok) eseményeket generálhatunk.

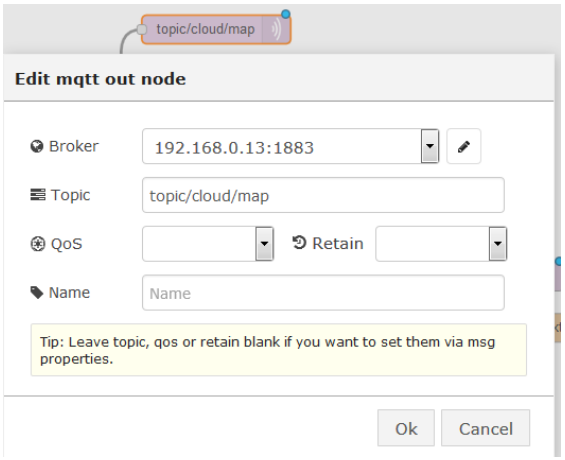
Ahhoz, hogy Node-RED-ben használni tudjuk az MQTT-t, szükségünk lesz egy MQTT brókerre, erre kivallóan alkalmas a mosquitto nevezetű program, ennek telepítéséről és konfigurációjáról a linken lehet többet megtudni.

Ha sikeresen feltelepítettük, akkor elegendő elindítani a mosquitto alkalmazást és a Node-RED-et is. Ezt követően elég a Node-RED-ben behúzni az MQTT node-okat és felkonfigurálni.

Beállítjuk a broker ip címét, továbbá input node esetén megmondjuk, hogy milyen topic-ba jött üzenetekre vagyunk kíváncsiak, outputnál pedig hogy melyik topic-ba akarunk posztolni.



11. ábra



12. ábra

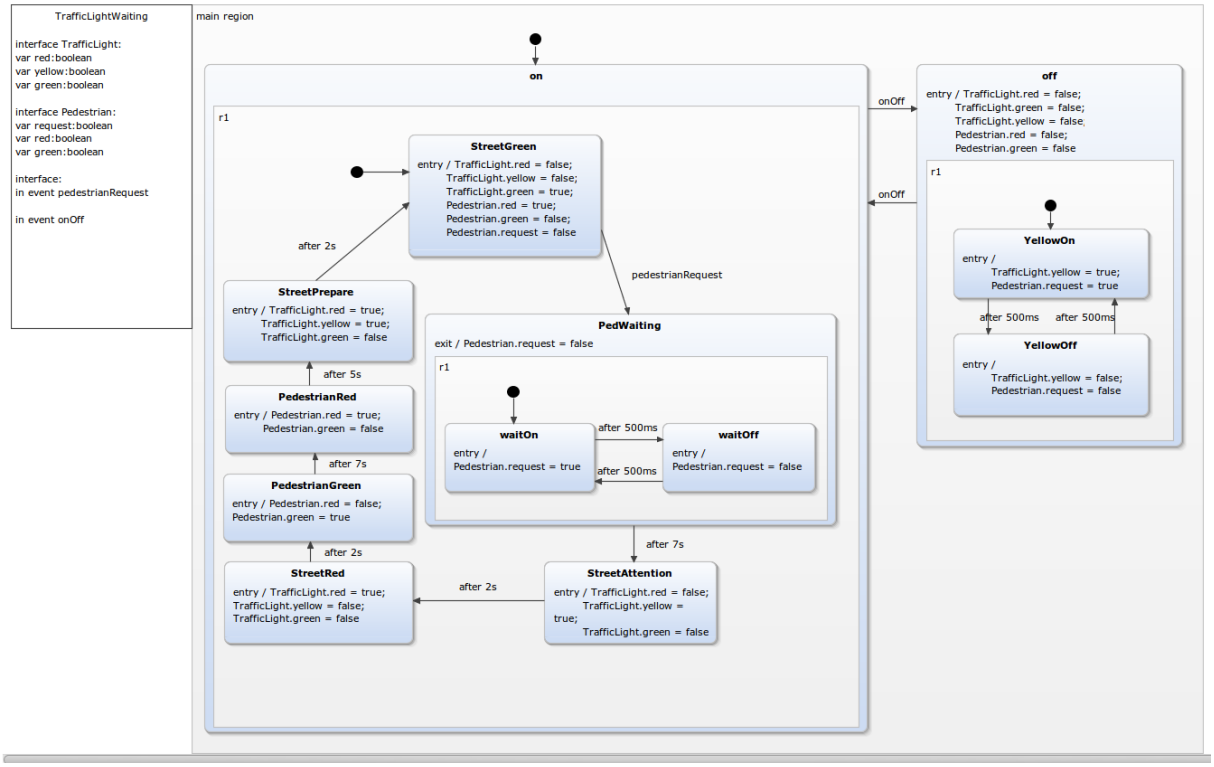
2.1.8 YAKINDU Statechart Tools 2

Egy ingyenesen elérhető, open-source eszköz, mely segítséget nyújt beágyazott rendszer modelljének mérnöki tervezésében és implementálásában. A nevéből adódóan ez egy eseményvezérelt állapotgép tervező környezet, mely grafikus programozó felületen könnyíti meg az állapotmodellek elkészítését. Az állapotmodellek az UML-ben meghatározott szabályoknak megfelelnek kisebb-nagyobb módosításokkal kiegészítve.

Négy fő alkotó eleme van a rendszernek. Az első a grafikus tervező felület, a második a Yakindu SCT 2 metamodell helyesség ellenőrző egység, a harmadik a szimulációs környezet, a negyedik pedig egy kódgenerátor, ami képes egy kattintással Java, C vagy C++ nyelveken implementálni az állapotmodelljeinket.

Az állapotgépek interfészeken keresztül képesek kommunikálni egymással, illetve a környezettel. Ezekben be-, illetve kimeneti eseményeket, változókat vagy függvényeket definiálhatunk.

A következő ábrán látható egy teljes modell, bal oldalon az interfészek definíciója, középen egy top-down tervezésű utcai jelzőlámpa állapot modelljének megvalósítása.



13. ábra YAKINDU SCT 2 közúti lámpa modell

3 A program

3.1 A feladat részletesebb leírása

A feladat egy beágyazott rendszer elosztott vezérlésének szimulálása cloud rendszer segítségével, és az ehhez kapcsolódó technológiák megismerése és ismertetése.

A beágyazott rendszerünket egy Raspberry Pi-al szimuláljuk. A Raspberry vezérlését egy Yakindu-ban elkészített állapot gépből generált C++ kód végzi. A robot rendelkezik szenzorokkal, amiknek segítségével meg tudja állapítani, hogy van-e akadály körülötte.

Az osztott vezérlést MQTT protokoll segítségével valósítjuk meg Node-RED-ben implementált logikával kiegészítve.

A Raspberry-n egy kivételesen buta robot mozgását szimuláljuk, hiszen a cél az volt, hogy minél előbb segítséget kérjen.

(Megj.: a robot logikáját nem lenne nagy erőfeszítés felokosítani, az állapotgépben csak minimális változtatást jelentene, így könnyen lehetne gyakorlati alkalmazást találni a project-nek. Például egy nagyon kis számítás igényű heurisztika futhatna a roboton a következő lépés irányának meghatározásához, és csak akkor kérne segítséget, ha bonyolult pálya lenne, ahol elakad, így tehermentesítve a server-t, ami akár sok ilyen robotot is kiszolgálhatna.)

A robot a szimuláció indulásakor lekéri az inicializálási adatait a szervergéptől, ezek a következők:

- a pálya térképe (ez csak a szimulációs helyzetben szükséges, hisz nincsen képünk a világról, amit a szenzorjaink érzékelhetnének, így a szenzorok szimulációjához szükség van a körülöttünk lévő világ adataira.) .
- A robot kiinduló pozíciója.
- A robot célpozíciója.

A robot buta lépés logikája úgy működik, hogy folyamatosan felfelé próbál lépni, ha nem sikerül, mert a szenzorjaival falat érzékel akkor pedig jobbra lép, ha ez is meghiúsul, akkor segítséget kér a szerver géptől majd vár a válaszára. Minden lépés után ellenőrzi, hogy elér-e a célba.

A szervergépen fut a MQTT broker és egy Node-RED alkalmazás, ami ha get üzenetet kap, akkor inicializáló információkat publish-ol, ha pedig help-et akkor egy útvonalat publish-ol a robot számára. Az útvonalat A* algoritmussal keresi meg.

3.2 A Node-RED alkalmazások

3.2.1 Saját Node

14. ábra

Ennek a node-nak az elkészítését én hajtottam végre felhasználva azt a kereső algoritmusokat megvalósító library-t, amit a forrásoknál feltüntettem.

A node-nak a feladata a szerver oldali logika megvalósítása.

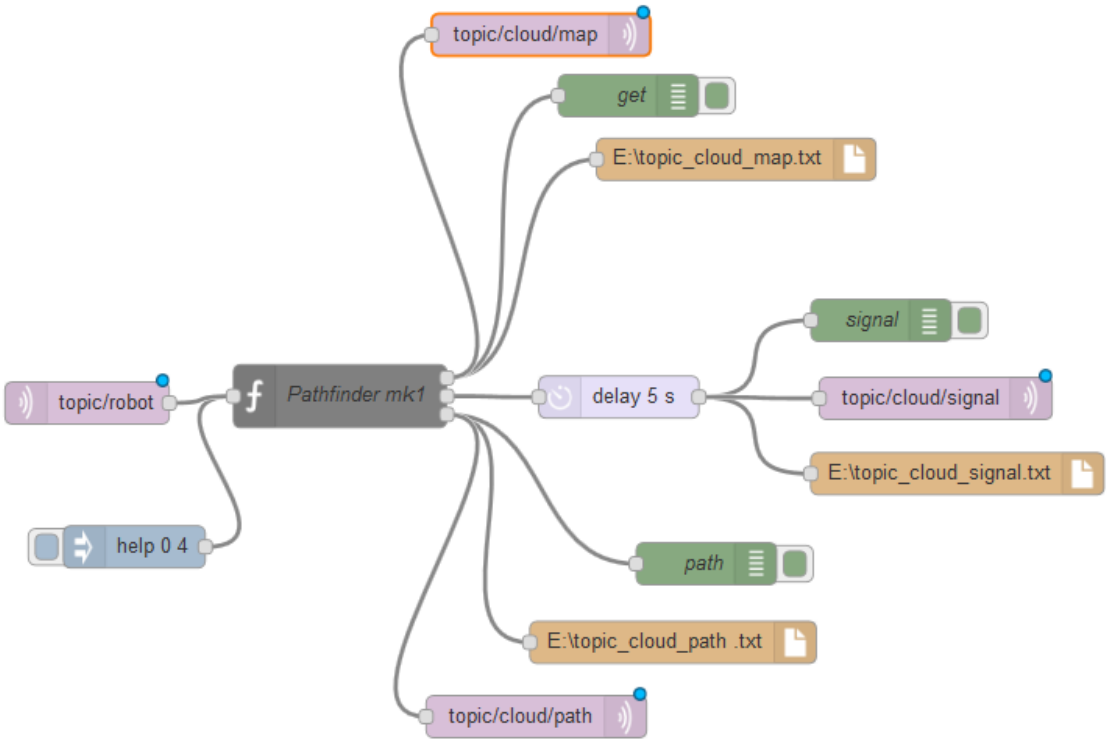
Bemenetének szerepe: A bemeneten kapja meg az üzenetet, ami a node működését határozza meg. Ez vagy egy get parancs lehet vagy egy help és koordináta parancs.

Kimenetek szerepe: Az 1. kimenetén az inicializáló adatokat küldi ki, a 2. kimenetén a jelzéseket, a 3. kimenetén pedig a végrehajtandó lépések irányát kódolva (0-fel, 1-jobbra, 2-le, 3-balra). (Megj.: kis bit bűvészkedéssel le lehetne optimalizálni úgy, hogy egy 8 bites integer-ben 4 lépést küldjünk el, ha a robot memóriája a szűk keresztmetszet).

Belső működés: Ha get parancs érkezik a msg.payload első szavaként, akkor inicializáló üzenetet küld ki az 1. kimenetén és egy jelzést a 2. kimenetén (1-es kód), a 3. kimenetre nem küld semmit. Az inicializáló üzenet egy mátrixot (térképet), egy kezdő pozíciót és egy célpozíciót tartalmaz. Ha help és két koordináta parancs jött akkor egy lépés sorozatot tartalmazó string-et küld ki a harmadik kimenetén és egy jelzést a második kimenetén (4-es kód), az első kimenetre nem küld semmit.

Keresőalgoritmus: A felhasznált kereső könyvtár nagyon sokféle algoritmus és heurisztika kombinációt támogat, amik közül csak egy egyszerű A*-ot használ a node, természetesen minimális módosítással tetszőleges algoritmusra át lehetne kapcsolni. Arra azonban figyelni kell, hogy nem mátrix indexeket használ, a pozíció definiálásához hanem (x,y) koordinátát, ami gyakorlatilag pont a fordítottja a megszokott i. sor, j. oszlop koordinátákhoz képest.

3.2.2 A szervergépen



15. ábra Szerver gépen futó debug verzió

3.2.2.1 A működése:

A ”topic/robot” topic-ba posztol a robot üzenetet. Ez az üzenet kétféle lehet, ami meghatározza a node működését:

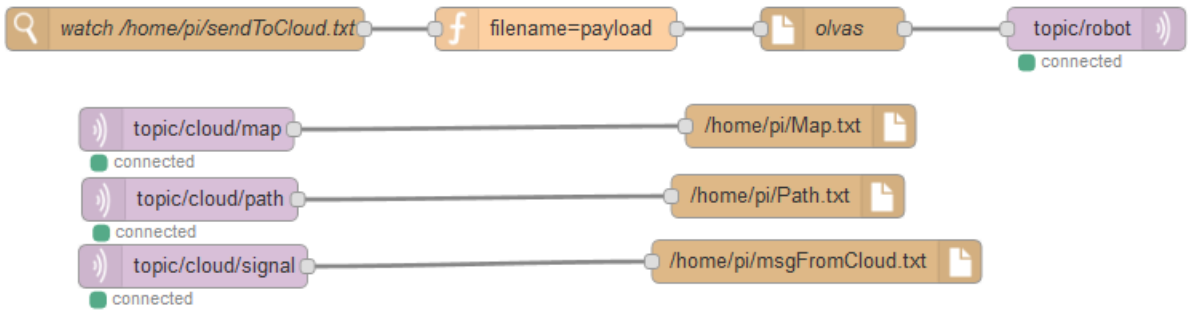
- Ha get-el kezdődik akkor a robot bejelentette igényét az inicializáláshoz szükséges adatok lekérésére. Ebben az esetben a pathfinder mk1-node megkapja, majd felismeri a kérést és kiküldi az inicializáláshoz szükséges mátrix-ot, kezdő pozíciót és vég pozíciót az 1. kimenetén. Ez az üzenet három node-ba torkollik, egy debug node-ba ami logolja az üzenetet a debug ablakba, egy File-ba, ami szintén logolást végez, továbbá az MQTT node-ba, ami publish-olja a topic/cloud/map topic-ba az inicializálási információt, a robot innen fog tájékozódni az adatairól. Ezt követően a második kimenetén, egy előzetesen definiált nyelvben meghatározott jelzést küld ki (esetünkben ez egy int (1)), hogy az inicializáló adatok elküldésre kerültek, ez a gyakorlatban egyszerre történik, de a delay node gondoskodik arról, hogy az adatok elküldése után küldje csak el a jelzést.

- Ha help-el kezdődik az üzenet, akkor a ”Pathfinder mk1” node úgy értelmezi, hogy segítséget kértek tőle. A msg.payload-ban a "help" után space-szel tagolva szerepelnie kell a robot (x,y) koordinátájának (int) (pl. ”help 4 0”), ha ez teljesül, akkor a node lefuttat egy A* algoritmust a robot által küldött koordinátáról indítva (a robot saját koordinátája) és a cél koordinátára. A célt és a pályát a szervergép ismeri, hisz ő inicializálta a robotot rá. A futás végeztével az útvonalat kódolja és a 3. kimenetén kiad egy msg-t aminek a payload-ja a lépések irány kódja ’,’-vel elválasztva pl.(”1,1,1,2,2,3,3,4,” ahol az 1=fel 2=jobbra 3=le 4=balra). Ezt az üzenetet szintén logoljuk debug window-n és file-ba is, továbbá publikáljuk a topic/cloud/path topic-ba, és egy másik jelzés kódot küldünk ki (’4’) a 2. kimenetre.

A jelzésekre a roboton futó c++ kód miatt van szükség, ezzel érjük el, hogy csak akkor próbálja meg elérni az útvonalat/inicializálási adatokat tartalmazó file-t, ha már benne van az adat.

Tesztelési célzattal van inject node a folyam elején, hogyha a robot nem is küld üzenetet, akkor tudjuk szimulálni.

3.2.3 A roboton



16. ábra

Működése:

Négy különálló részből áll:

- A …/path topic-ba érkező publish-okat beleírja a path.txt file-ba, innen fogja a c++ kód feldolgozni az útvonalat.

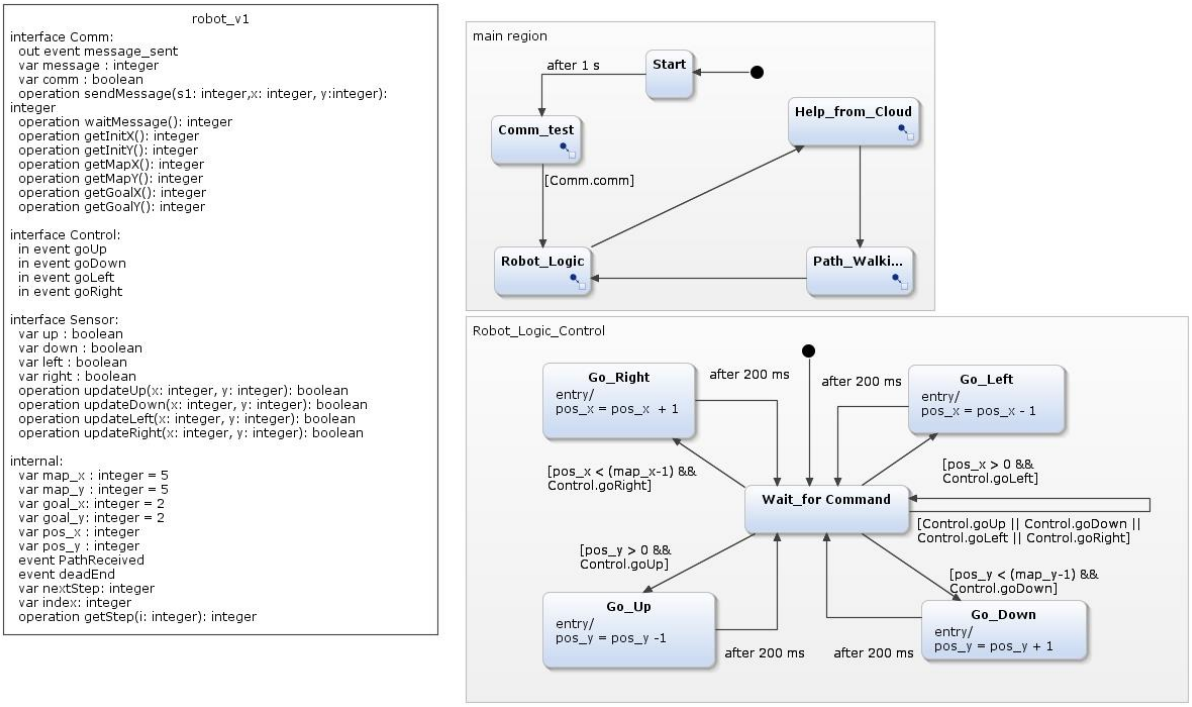
- A …/map topic-ba érkező publish-okat beleírja a map.txt file-ba, innen fogja a c++ kód feldolgozni az inicializációs adatokat.

- A …/signal topic-ba érkező publish-okat beleírja a msgFromCloud.txt file-ba, c++ kód ezen keresztül figyeli, hogy jött-e üzenet, és a benne található kód alapján, dönti el, hogy melyik file-t kell beolvasnia, a Map.txt vagy a Path.txt.

- A watch node figyeli, hogy a megadott fileokat írták-e, ha igen, akkor generál egy msg-t ami payload-jában a file nevét tárolja. Mivel a file-ból olvasó node csak a filename property-ben képes átvenni elérési útvonalat, ezért egy függvényt kell beiktatni a két node közé, ami a msg.filename=msg.payload; utasítást futtatja, és így feldolgozhatóvá teszi az adatot a ''file'' node számára. A file node ezt követően kiolvassa az adatokat (jelzéseket (get, help)) a sendToCloud.txt-ből és publish-olja a topic/robot-ra amin keresztül a szerver értesül a robot igényeiről.

3.3 A Yakindu állapotmodell

Az állapotgép modell tervezését top-down módszerrel kezdtem el, mellyel a következő állapotmodell állt elő.



17. ábra YAKINDU SCT 2 állapotmodell a feladat megoldásához

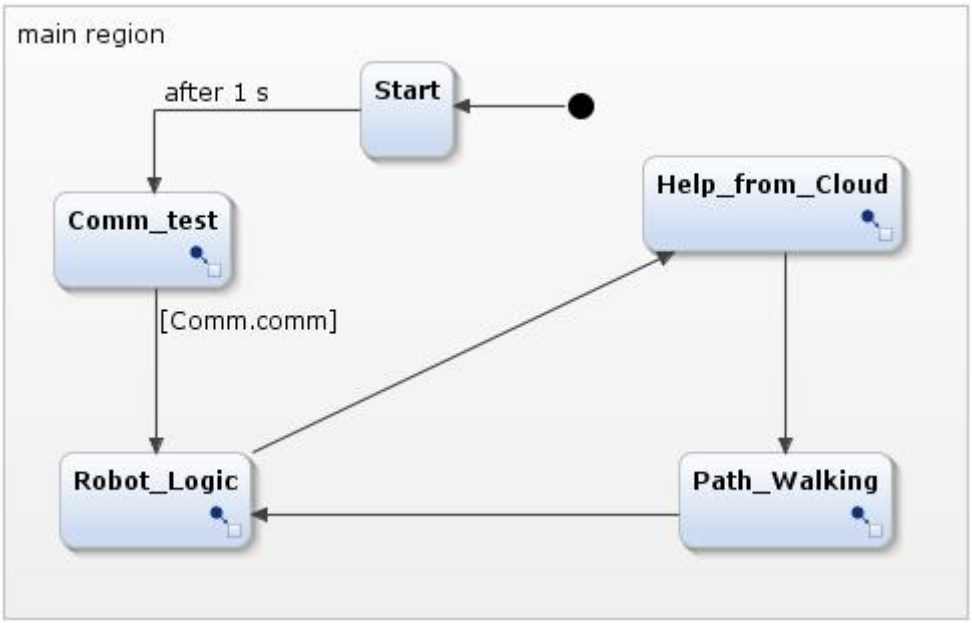
3.3.1 A robot_v1 projekt interfészei

- Comm: Az üzenet küldésért és fogadásért felelős a robot és a cloud rendszer közt.
  - message_sent: Amikor üzenetet küld a cloud felé ezt az eseményt tüzeli el a robot.
  - message: Az üzenet kódját tartalmazza, amit utoljára kapott a robot a cloud-tól.
  - comm: Egy boolean változó, értéke igaz, ha sikerült kapcsolatot létesítenie a cloud-dal.
  - integer sendMessage (s1: integer, x: integer, y_integer): Eljárás, ami elküldte egy s1 értékű üzenetet a robot jelenlegi pozíciójával együtt.
  - integer waitMessage (): Eljárás, amely visszatér az utolsó üzenettel, amit a cloud küldött.
  - integer getInitX (), integer getInitY (): Eljárások, amelyek megadják a robot kezdőpozícióját a pályán.
  - integer getMapX (), integer getMapY (): Eljárások, amelyek megadják a játéktér két dimenziójának maximumát.
  - integer getGoalX (), integer getGoalY (): Eljárások, amelyek megadják a célpozícióját a robotnak.
- Control: A robot helyváltoztatásáért felelős interfész, segítségével direktben tudjuk irányítani a robotot.
- Sensor: A robot körül található négy szenzor, amelyekkel érzékeli a külvilág akadályait.
  - boolean update (Up/Down/Left/Right)(x: integer, y: integer): Eljárások alkalmasak lekérdezni a megfelelő szenzor adatait.
- internál: Ebben az interfészben tárolja a külvilág számára nem elérhető változókat, függvényeket, eseményeket.
  - integer map_x, map_y: Tárolja a játéktér két dimenziójának szélső értékét.
  - integer goal_x, goal_y: Tárolja a célpozíciót.

- integer pos_x, pos_y: Tárolja az aktuális pozíciót.
- event PathReceived: Esemény, mely akkor sül el, ha útvonal adatot kap a robot a cloud-tól.
- event deadEnd: Esemény, mely akkor sül el, ha zsákutcába jut a robot.
- integer nextStep: Tárolja a következő lépésnek megfelelő irányt, ami a cloud-tól kapott útvonal egy eleme.
- integer index: Egy szimpla ciklusváltozó, amely a cloud-tól kapott útvonalat járja be egyesével.
- integer getStep (i: integer): A függvényparaméterként átadott sorszámú lépéssel tér vissza.

3.3.2 Állapotmodell tervezése

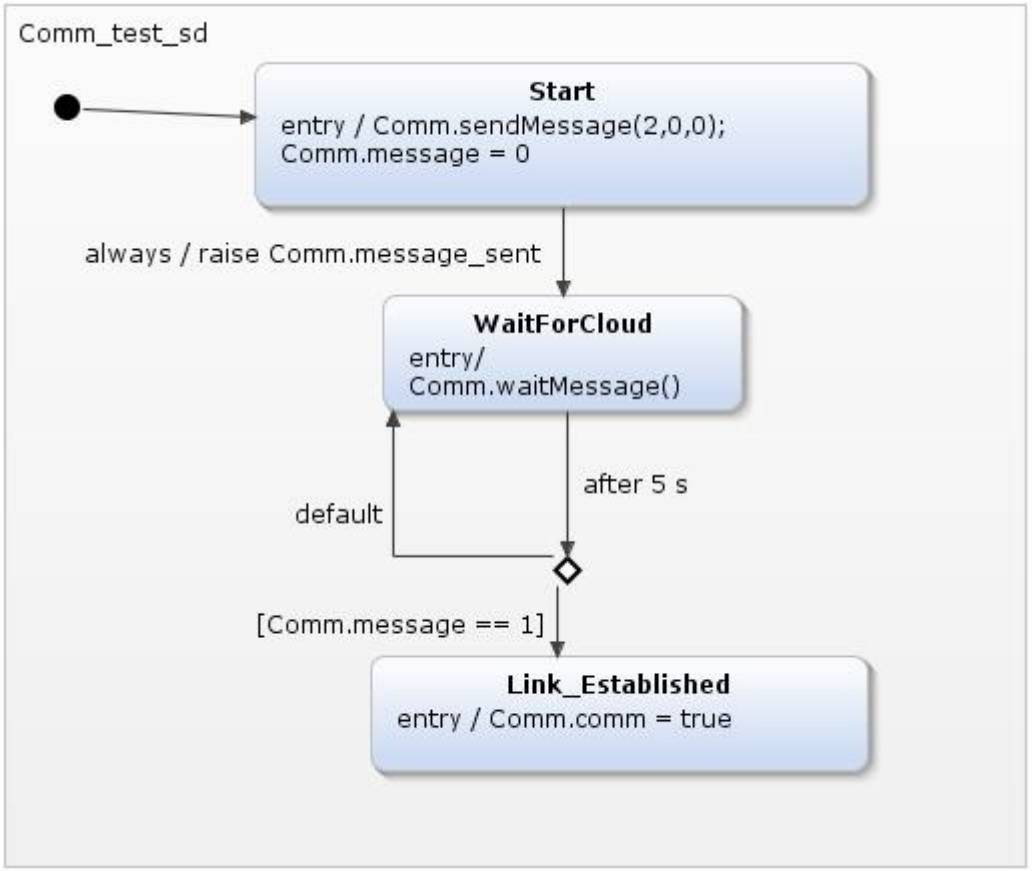
A következőekben ismertetem az állapotmodell elemeket.



18. ábra Top szint

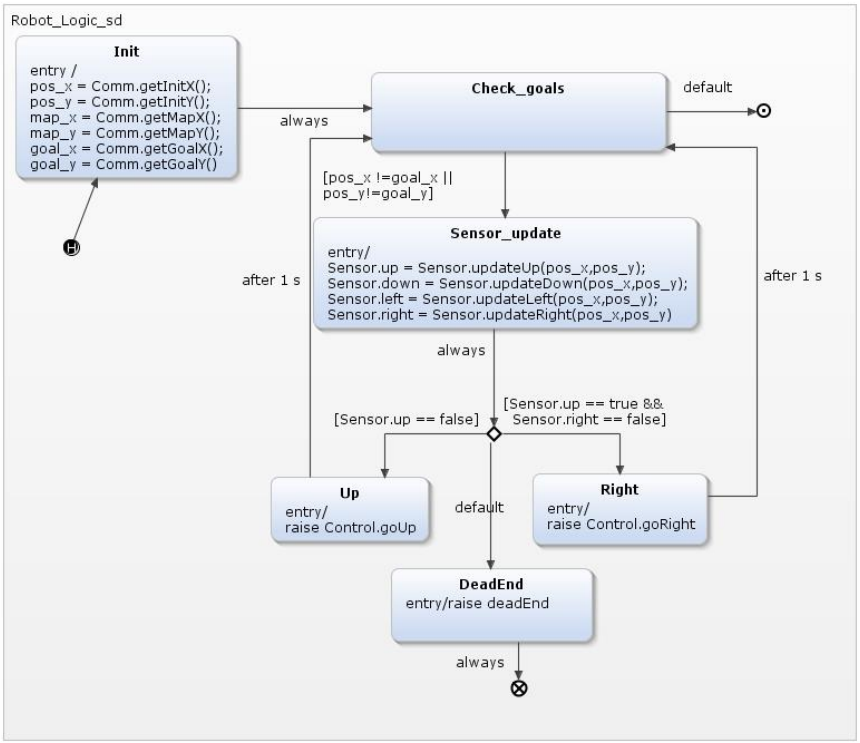
A top szint nagyon magas absztrakciós szinten fogja meg a problémát. Szavakkal leírva ez a folyamat:

- Elindul a robot, egy másodpercet vár, hogy minden fontos komponens inicializálódjon. A kommunikációt teszteli a cloud egységgel, csak akkor lép tovább, ha megbizonyosodott a cloud létezéséről és készen állásáról. Majd megkezdi a futását. Optimális esetben a robot logika megoldja a problémát, eljut a célállapotba. Ha ez nem sikerül akkor a cloud-hoz fordul, ami elküldi számára a bejárandó utat, hogy eljusson céljába. Ezt az útvonalat bejárja és visszatér a problémamegoldó fázisba. Ott fogja vizsgálni a cél pozíció elérését. Ha elérte a célját terminálja a működését a robot.



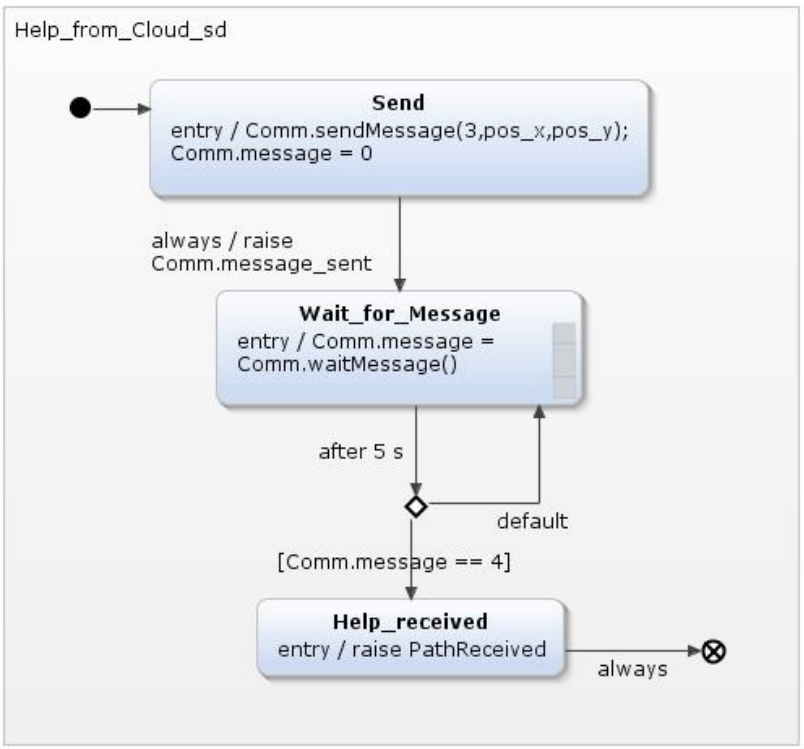
19. ábra Comm_test subdiagram

A comm_test subdiagram-ban a robot kezdeményez kapcsolatot a cloud-dal, majd addig várakozik, amíg a cloud vissza nem jelez, hogy kész a segítség nyújtásra.



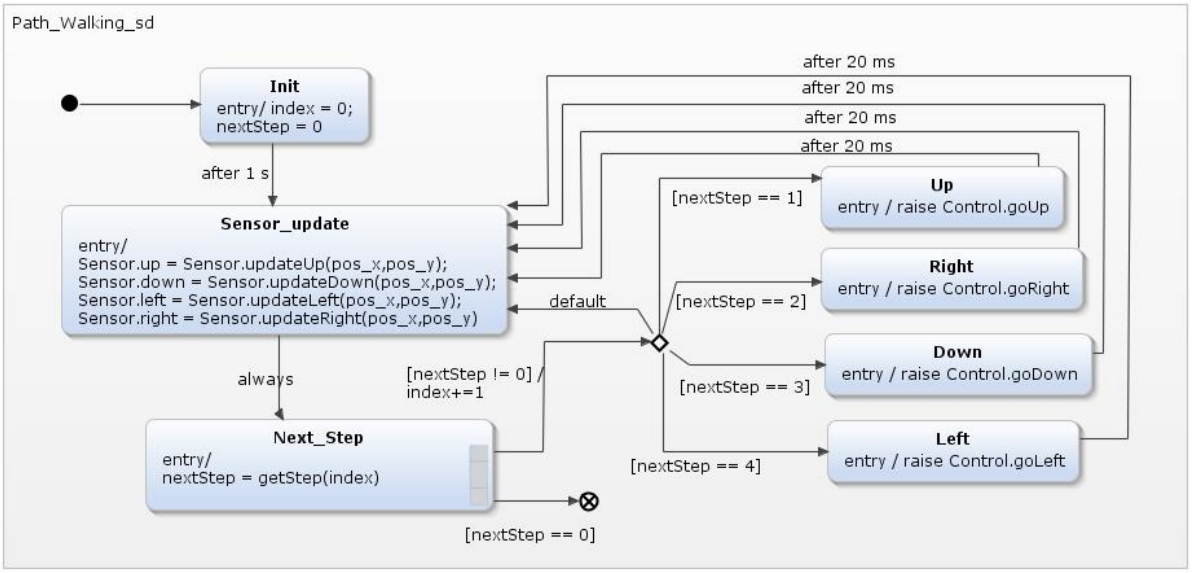
20. ábra A Robot_Logic subdiagram

A Robot_Logic subdiagram felelős a robot probléma megoldó logikájának megvalósításáért.



21. ábra Help_from_Cloud subdiagram

Help_from_Cloud subdiagram a Comm_test-hez hasonlóan üzenetet küld a cloud-nak, majd megvárja míg válaszol az.



22. ábra Path_Walking subdiagram

Path_Walking subdiagram végrehajtja a kapott útvonal bejárásához szükséges parancsokat.

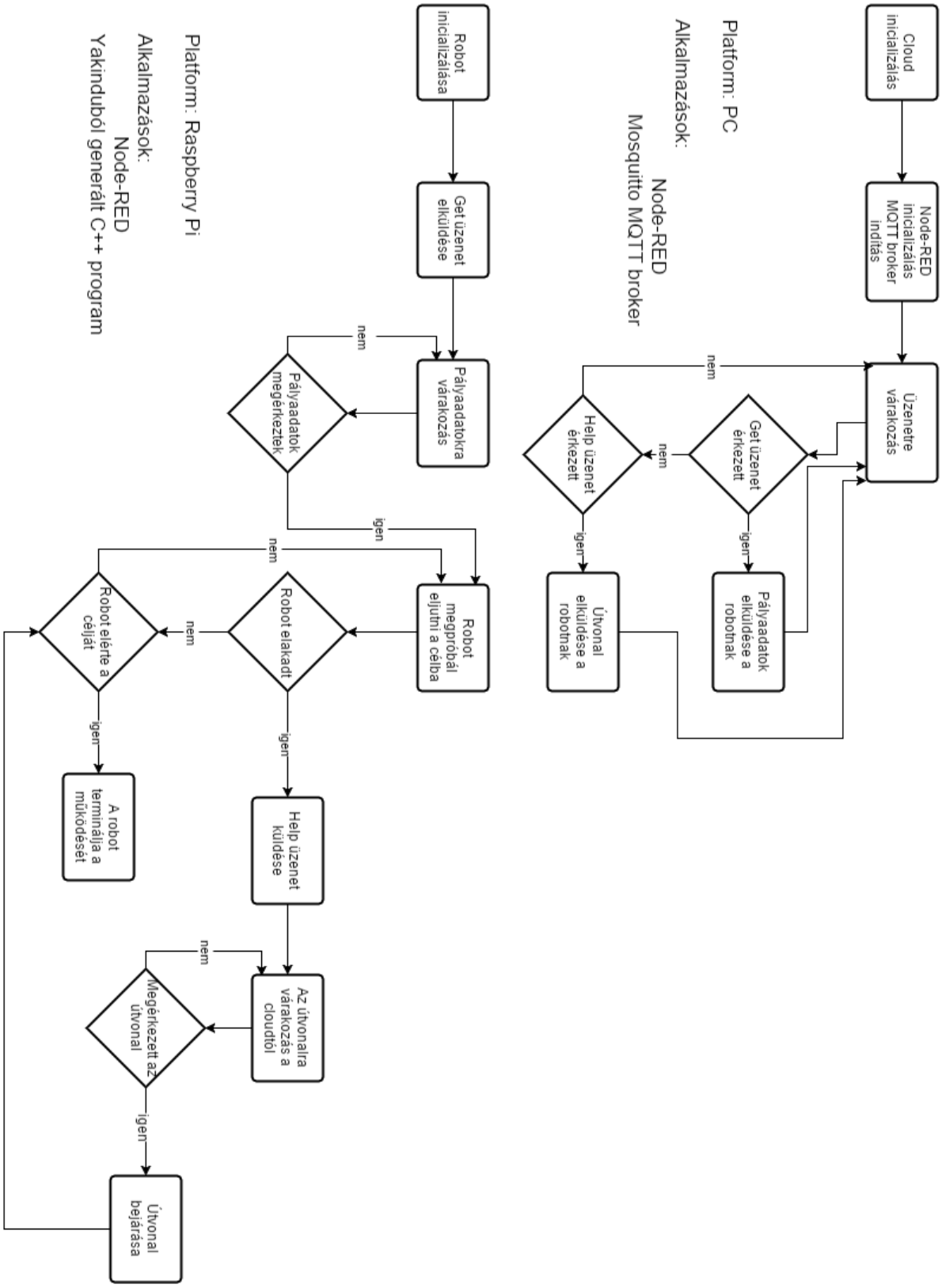
3.3.3 Kódgenerálás

A YAKINDU SCT 2 egyik komponense, a kódgenerátor. Sikeresen generáltam kódot az állapotmodellből, amit azután lefuttattunk.

Az állapotmodellt kiegészítettem egy C++11 szabványban íródott függvényekkel és szálkezelt időzítő osztállyal, amik létrehozzák a kommunikációt a robot és a cloud közt és biztosítja az állapotgép helyes működését.

4 Áttekintés

A következő oldalon folyamatábra (23.ábra) szemlélteti a dolgozat megoldását egy magas absztrakciós szinten.



23. ábra A megoldás folyamatábrával szemléltetve

Hivatkozások

[1] https://www.npmjs.com/package/pathfinding

[2] http://www.rs-online.com/designspark/electronics/eng/blog/building-distributed-node-red-applications-with-mqtt

[3] http://nodered.org/

[4] http://statecharts.org/documentation.html