

Felhő alapú elosztott vezérlés, yakinduban tervezett állapotgépet használó beágyazott robot rendszeren.

Készítette

- Kovács Levente Ákos - CM6UKU
- Tóth Krisztián Dávid - J38GIK

Tartalomjegyzék

Bevezetés.....	3
A feladat:	3
Program futása:	3
Felhasznált Technológiák:	3
A cloud rendszerhez:	3
A robothoz:	3
Technológiák:	4
Node-red	4
Bevezetés:	4
Telepítése:	4
Nodeok:	5
Nodeok készítése/felépítése:	8
Message kezelés:	12
Mqtt És a Node Red.....	13
Yakindu: TODO	13
A program:.....	14
A feladat részletesebb leírása:	14
A node-red alkalmazások:	15
Saját Node:	15
A szervergépen	16
A roboton	17
A Yakindu alkalmazások: TODO.....	18

Bevezetés

A feladat: Egy labirintusban közlekedő buta(magától csak jobbra és felfele közlekedő) robot szimulálása beágyazott rendszeren(Raspberry Pi), ami adott kezdeti állapotból egy vég állapotba próbál eljutni, egy pc-n futó cloud rendszer segítségével. Továbbá az ehhez tartozó technológiák megismerése(Node-red, Yakinidu).

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(szenzorai falat érzékelnek mind 2 irányban), 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* út kereső algoritmust a robot és a cél koordinátaival. Majd vissza küldi a robotnak. A gyakorlati megvalósításokról külön fejezetekben részletesebben írunk.

Felhasznált Technológiák:

A cloud rendszerhez:

A kommunikációs logika Node Red-ben került implementálásra MQTT (Message Queueing Telemetry Transport) protokoll felhasználásával, amit a mosquitto(mqtt broker) valósít meg. A kereső motornak IMOR nevű felhasználó pathfindig Node package-ét használja.

A robothoz:

Yakiniduban tervezett állapotgépből generált C++ kódot futtat, továbbá szintén node red fut a kommunikáláshoz.

Technológiák:

Node-red

Bevezetés:

A Node-RED egy grafikus felület hardwarek-ek, Apik, és online szolgáltatások összekötésére, és az esemény vezérelt 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(1 parncs cmd-ből), továbbá optimálissá teszi cloudon és raspberry Pi on való felhasználásra.

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

Egy flow kialakítását web browser-ben(firefox,chrome...) a node red portjára(sajátip: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(munkalap)-ek kiválasztásával tudunk váltogatni, és itt tudunk újat 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.

Telepítése:

(Megj: Jól érthető angol leírás található a 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 nodered-et, a node Package manager(npm)-en keresztül az alábbi paranccsal:

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


Ez a parancs a C:\Users\„felhasználó”\AppData\Roaming\npm mappába fogja globálisan telepíteni a node-red-et , ami után bárholnan futtatni lehet parancssorból a node-red parancs kiadásával.

Nodeok:


A node-ok, a folyamainknak(flow) az építő elemei. Ezekből huzalozzuk össze a grafikus felületen a megvalósítandó rendszerünket. A node-ok működését úgy tudjuk beállítani hogy 2x klikkelünk a modul ikonjára és a felugró ablakon beállítjuk a megfelelő konfigurációs értékeket.

PL: Inject node beállítása


Edit inject node

 Payload

timestamp

 Topic

Időbélyeg óránként 6-12:00ig pénteken

 Repeat

interval between times

every

60

 minutes

between

06:00

 and

12:00

on

☐ Monday

☐ Tuesday


☐ Wednesday

☐ Thursday

☒ Friday

☐ Saturday

☐ Sunday

 Name

Test

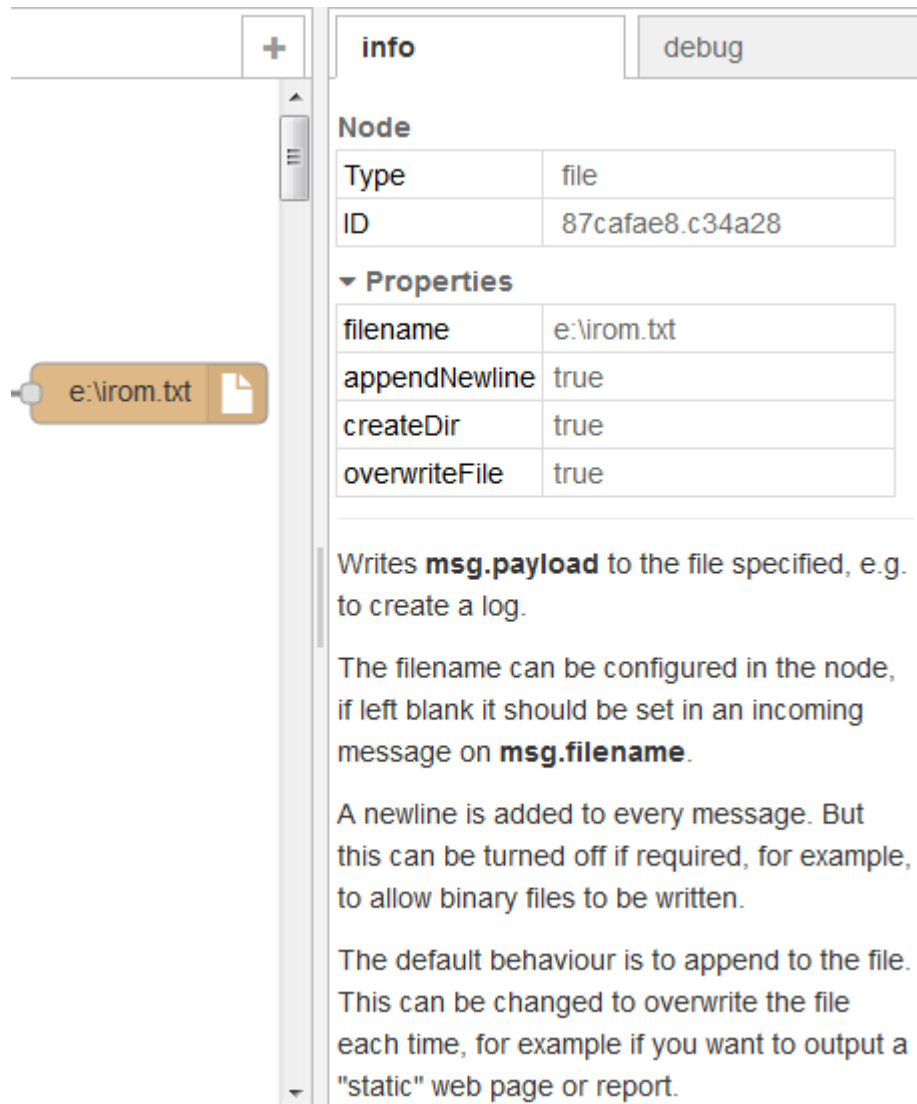
Note: "interval between times" and "at a specific time" will use cron.
See info box for details.

Ok

Cancel

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

PL: file(iró) node info palettája.



The screenshot shows the Node-RED interface with the 'info' tab selected for a 'file' node. The node is named 'e:\irom.txt'. The 'info' tab displays the following information:

Node	
Type	file
ID	87cafae8.c34a28

Below the node information, the 'Properties' section is expanded, showing the following properties:

filename	e:\irom.txt
appendNewline	true
createDir	true
overwriteFile	true

At the bottom of the 'info' tab, there is a description of the node's function:

Writes **msg.payload** to the file specified, e.g. to create a log.

The filename can be configured in the node, if left blank it should be set in an incoming message on **msg.filename**.

A newline is added to every message. But this can be turned off if required, for example, to allow binary files to be written.

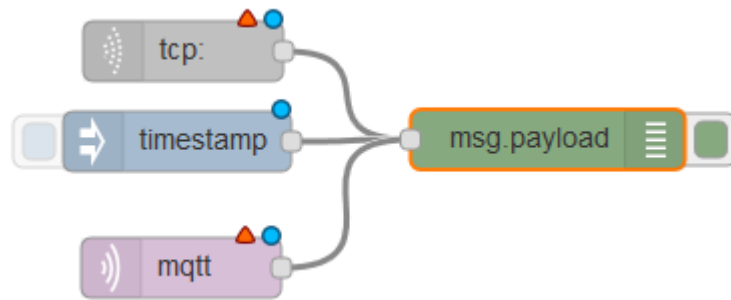
The default behaviour is to append to the file. This can be changed to overwrite the file each time, for example if you want to output a "static" web page or report.

Megkülönböztethetjük őket :

Funkció szerint:

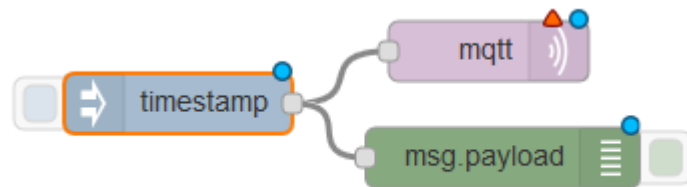
- Input(Bemeneti): Esemény generálásra használatos, ez lehet különböző protokollok-ról (Tcp/Udp/Mqtt) beérkező üzenet-ből létrehozott node msg vagy csak egyszerű esemény injektálás. Az esemény injektáló nodeot be lehet úgy állítani, hogy ismétlje az esemény injektálást bizonyos időközönként (pl mp-enként), 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.



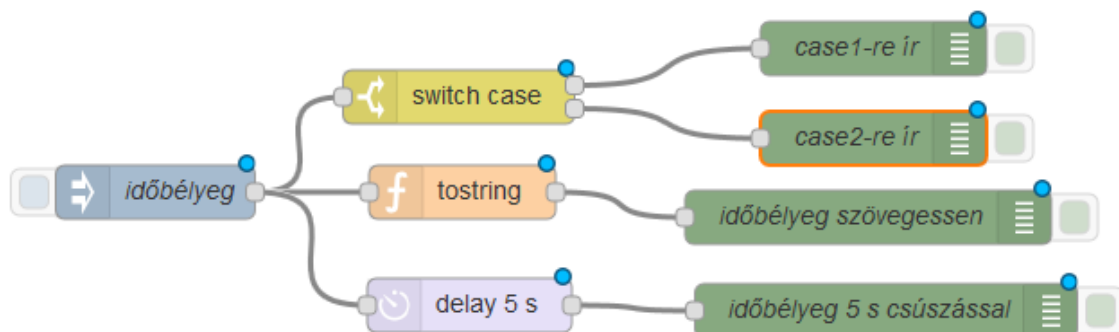
(Bejövő üzenetek payloadját kiírja a debug ablakban)

- Output(Kimeneti): Csak torkollanak bele folyamatok, ki nem indulnak belőle. Főleg üzenet továbbításra használjuk, pl udp,mqtt csomag küldése vagy debug node-ot(zöld) a message payload-gyának logolásra.



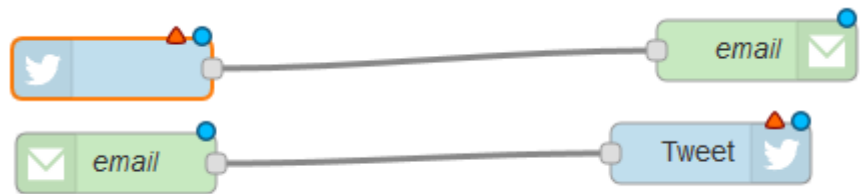
Időbélyeget küld ki mqtt-n keresztül, továbbá kiírja a debug ablakba.

- Function(függvény):Olyan nodeok amik valamilyen adatmanipulációt (xml,Json convert to/from ...), késleltetést(delay), vagy valamilyen kapcsolat felvételt(tcp/http request) hajtanak végre. Ezeknek a nodeoknak tipikusan kimenetet 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 általt Java scriptben megírt kód futhat benne. Ez főleg egyszerű de egyedi adatmanipulációra jó, de a komplexebb feladatokhoz érdemes saját nodeot készíteni (Isd később).

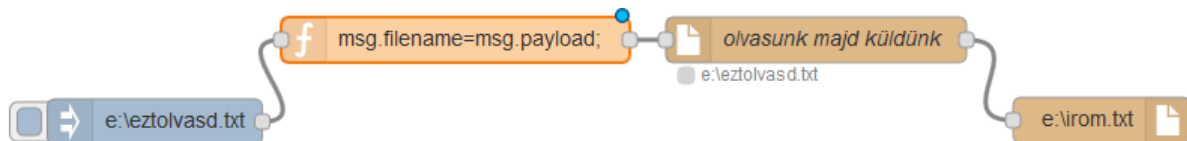


- Social: Twitter/Email üzenetek lekérésére és küldésére használt nodeok.

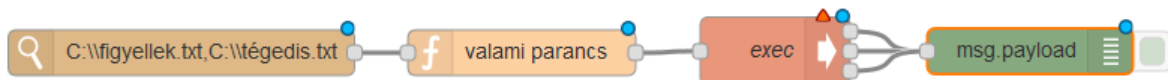
(tweetekből emailt-küld , emaileket tweeteli)



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



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



Kimenetek/bemenetek száma szerint:


Egy nodenak lehet tetszőleges számú kimeneti és 0 vagy 1 bemeneti socket-je is (ezek kezeléséről ld később). Egy bemenet socket-be tetszőleges számú event folyhat be (pl : Advanced példa-nál execből mind a 3 kimeneti socket a debug 1 bemenetével van összekapcsolva), és egy kimeneti socket-ből akár több irányba is továbbíthatunk egyszerre msg-t (pl Output példánál). Több kimeneti socket-el, külön választhatjuk az adatfolyamunkat több párhuzamos végrehajtási irányban, vagy feltételhez köthetjük az üzenet útját/tartalmát .

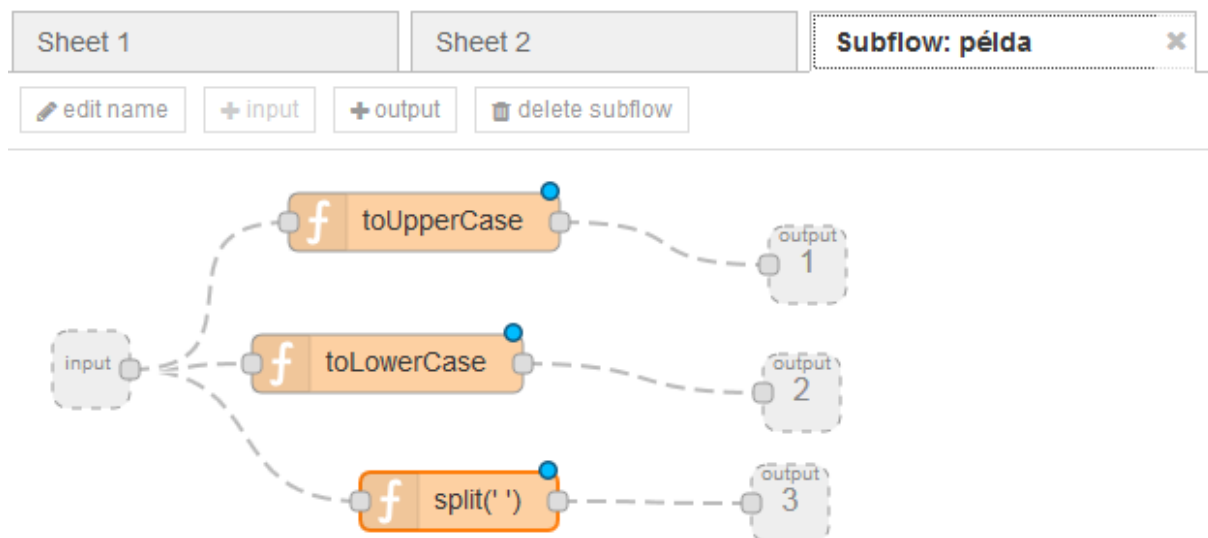
Nodeok készítése/felépítése:

Egy node-ot 3 féle képen lehet létrehozni,

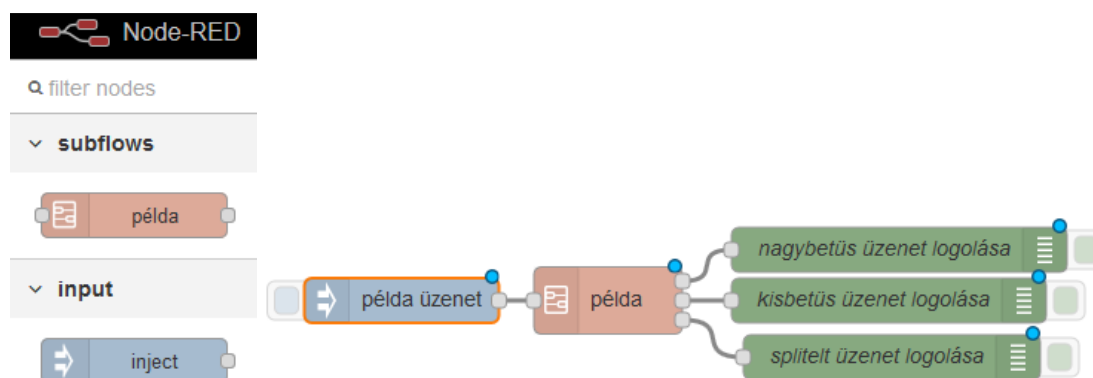
1. **Internetről npm-en keresztül :** Ez a legegyszerűbb módszer, keresünk az npmjs.com oldalon található node-ok közül egy olyat amire nekünk szükségünk van majd kiadjuk a `npm install node-red-node-{filename}` parancsot és telepítjük (Megj:minden node red node neve node-red-node-* al kezdődik, a többi sima node.js package ezeket a 3. módszerben ismertetett módon lehet felhasználni.)

2. Subflow-ként:

Lehetőségünk van a meglévő node-jaink-ból 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. És kialakíthatunk egy flow-t a hozzátartozó logikával. PL:



Ha ezzel készen vagyunk meg fog jelenni a node-listában a subflow menü pont alatt az új nodeunk. Innentől pedig használhatjuk, csak be kell huzni egy flowba.



3. Manuálisan (Saját node készítése/nodeok felépítése):

Minden node 2 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 (elég hozzáértéssel), a settings.js file manipulálásával (ami a node red könyvtárban található). (Megj: A nodeok készítéséről jó leírás és példa található a node-red weboldalon)

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

pl (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>
```

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 készre 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 hívodik meg mikor valamilyen 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);  
}
```

Message kezelés:

Node red-ben message-ekkel kommunikálnak a node-ok, ezek JSON objectumok 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 nodeok amik több property-t is használnak, ilyen például a file („in”) node ami ha nem égetve kapja meg az olvasni kívánt file elérési utvonalát, akkor a msg.filename property-től várja ezt. De a saját nodejainkban is tetszőleges számú property-t bevezethetünk. (Megj: a msg-k 2 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)

Példa kódok üzenet manipulációra:

- Üzenet tovább küldése:

```
return msg;
```

- Új üzenet létrehozása és továbbküldése:

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

- Üzenet szétválasztása (2 output közül csak az egyiket továbbítja az üzenetet a másikon nem (nem küld NULL üzenetet, amelyik outputra null megy azon megszakítja a folyamatot))

```
if (msg.topic == "feltétel") {  
    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 szemlélítve:

```
var msg1 = { payload: "első kimenete az output 1-nek" };  
var msg2 = { payload: "második kimenete az output 1-nek" };  
var msg3 = { payload: "harmadik kimenete az output 1-nek" };  
var msg4 = { payload: "egyetlen üzenete 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 világ" };  
node.send(msg1);
```

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:

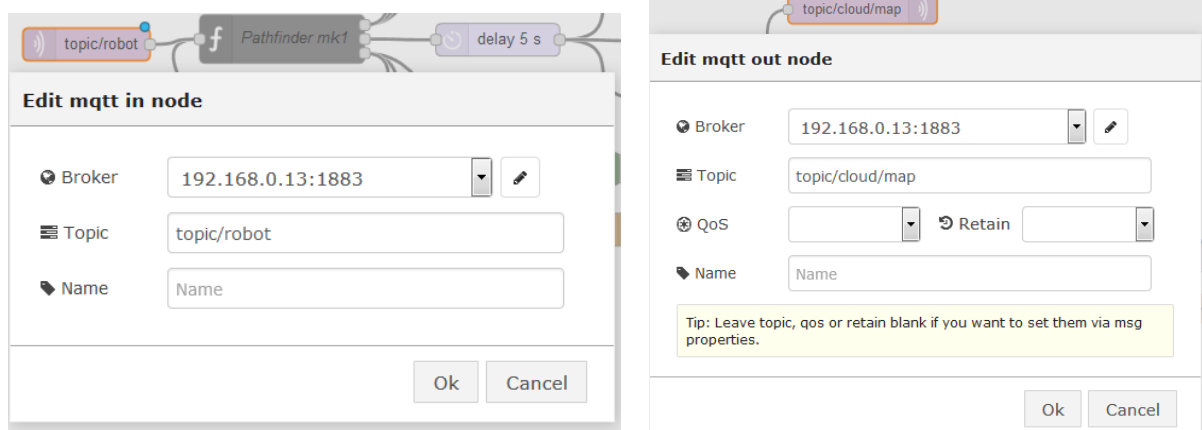
<http://www.rs-online.com/designspark/electronics/eng/blog/building-distributed-node-red-applications-with-mqtt>

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 kiválóan 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-redet is. Ezt követően elég a node red-ben behúzni az mqtt-t node-okat és felkonfigurálni.

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



Yakindu: TODO

A program:

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 servert 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 szenzoraink 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 szenzorival 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 szervergép-en fut a mqtt broker és egy node-red alkalmazás ami, ha get üzenetet kap akkor inicializáló információkat publish-ol, ha pedig helpet akkor egy utvonalat publishol a robot számára. Az utvonalat A* algoritmussal keresi meg.

A node-red alkalmazások:

Saját Node:



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+koordináta parancs.

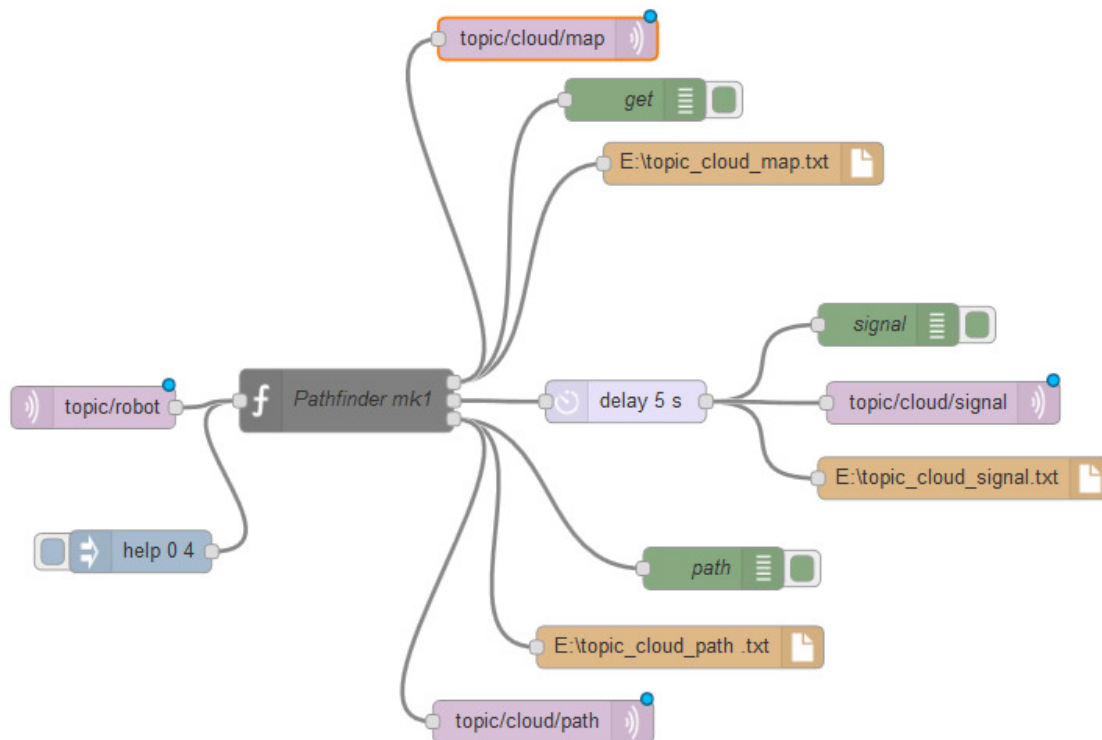
Kimenetek szerepe: Az 1. kimenetén az inicializáló adatokat küldi ki, a 2. kimenetén a jelzéseket, a 3. kimenetén pedig a végrehajtandó lépések irányát kódolva (0-fel,1-jobbra,2-le,3-balra).(Megj: kis bit bűvészkedéssel le lehetne optimalizálni úgy hogy egy 8 bites int-ben 4lépést küldjünk el, ha a robot 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+2 koordináta parancs jött akkor egy lépés sorozatot tartalmazó stringet küld ki az 3. kimenetén és egy jelzést a 2.kimenetén(4-es kód), az 1. 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.

A szervergépen az alábbi node-red modell fut :

(Debug verzió)



A működése:

A "topic/robot" topicba posztol a robot üzenetet. Ez az üzenet 2 féle lehet, ami meghatározza a node működését:

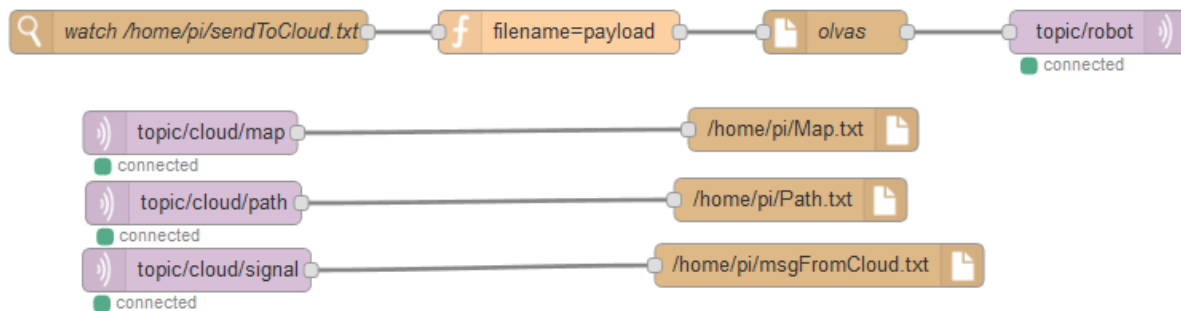
1. Ha **get**-el kezdődik akkor a robot bejelentette igényét az inicializáláshoz szükséges adatok lekérésére. Ebben az esetben a pathfinder mk1-node megkapja majd felismeri a kérést és kiküldi az inicializáláshoz szükséges mátrix-ot , kezdő pozíciót és vég pozíciót az 1. kimenetén. Ez az üzenet 3 node-ba torkollik, egy debug nodeba ami logolja az üzenetet a debug ablakba, egy File-ba ami szintén loggolást végez, továbbá az mqtt node ba, ami publish-olja a topic/cloud/map topicba az inicializálási információt, a robot innen fog tályékozodni az adatairól. Ezt követően a 2. 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.
2. 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-el 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 cél-t és a pályát a szervergép ismeri, hisz ő

inicializálta a robotot rá. A futás végeztével a utvonalat 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 utvonalat/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 mégis tudjuk szimulálni.

A roboton futó node-red modell:



Működése:

4 különálló részből áll:

1. A .../path topicba érkező publish-okat beleírja a path.txt file-ba, innen fogja a c++ kód feldolgozni az utvonalat.
2. A .../map topicba érkező publish-okat beleírja a map.txt file-ba, innen fogja a c++ kód feldolgozni az inicializációs adatokat.
3. A .../signal topicba érkező publish-okat beleírja a msgFromCloud.txt file-ba, c++ kód ezen keresztül figyel, 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.
4. A watch node figyel, hogy a megadott fileokat irták-e, ha igen, akkor generál egy msg-t ami payloadjában a file nevét tárolja. Mivel a file-ból olvasó node csak a filename propertyben képes átvenni elérési útvonalat, ezért egy függvényt kell beiktatni a 2 node közé, ami a `msg.filename=msg.payload;` utasítást végrehajta, 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.

A Yajindu alkalmazások: TODO

Források:

<https://www.npmjs.com/package/pathfinding>

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

<http://nodered.org/>