Système de monitoring batteries pour bateau

Prévu pour surveiller la tension et le courant de charge/décharge de 3 batteries 12V

Jérémy LANIEL et Yann HELLO
www.kp3d.re

Explication fonctionnement logiciel

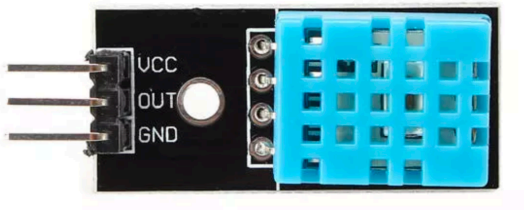
(merci d’être tolérant, je ne suis pas informaticien et donc je n’ai pas forcément ni le langage adapté ni la bonne façon d’expliquer tout le temps 🤭)

Matériel

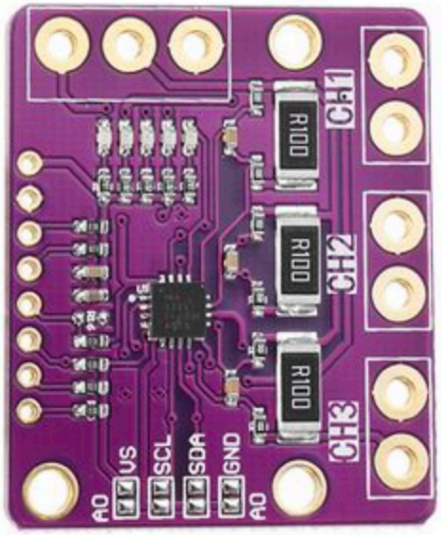
Le cœur de ce système est l'ESP8266. C'est un microcontrôleur qui a par construction des fonctions pré-programmées pour les liaisons WiFi.
Il se programme de façon assez simple en langage Arduino avec son IDE dès l'instant qu'on a bien paramétré la cible ESP8266 (ici via la carte WEMOS D1).
Ce microcontrôleur est équipé d'entrées/sorties numériques et d'un convertisseur A/N 10 bits (attention la tension d'entrée ne doit pas dépasser 3,3V). Ces interfaces sont programmables de la même façon qu'un Arduino.
Il a aussi, et c'est ça qui est intéressant, une mémoire flash (en plus de la mémoire programme) dont la taille est configurable et s'utilise pour y mettre des fichiers (SPIFFS). La taille peut être de 1, 2 ou 3Mo. C'est là qu'on stockera les fichiers nécessaires à la description du serveur web (html, css, javascript,…).
Nous verrons que cette taille est très largement suffisante pour y mettre un serveur web relativement complet et bien plus complexe que celui que je propose dans ce projet (il y a donc des améliorations possibles pour rendre cette interface utilisateur bien plus complexe et plus jolie)

I/F avec les périphériques

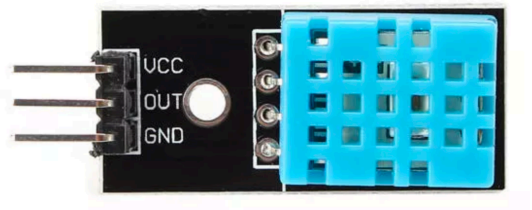
La première chose qu'on fait avec ce microcontrôleur c'est de récupérer les données de nos capteurs. On en a deux : le capteur de température (DHT11)



et le capteur de courants/tensions (INA3221).



DHT11

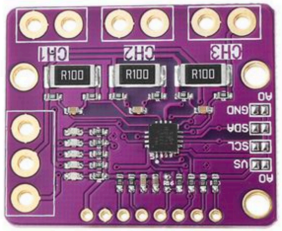


Le DHT11 a une interface numérique du type série propriétaire mais on trouve facilement les bibliothèques de programmes pour l'interfacer sur internet : « DHTesp.h »
Les deux variables où sont stockées les données acquises sont Temperaturebat (température) Rhbat (humidité).
On configure la lecture du DTH par « dht.setup(DHT11_PIN, DHTesp::DHT11); » et on vient lire les données par le morceau de programme :

```
delay(dht.getMinimumSamplingPeriod());
   if(isnan(dht.getTemperature())) Temperaturebat = 0;
   else {
   Temperaturebat = dht.getTemperature();
   RHbat = dht.getHumidity();
   }
```

Le « IF » n'est là que pour virer les fausses acquisitions de températures qui arrivaient dès fois sans trop que je comprenne pourquoi.

INA3221



Le INA3221 a une interface numérique du type I2C (un bus en fait). Cette interface est très pratique puisqu'on peut mettre plein d'objets de ce type sur le même bus, chaque objet ayant sa propre adresse. C'est grâce à cette fonction qu'un jour j'ajouterai un module I2C qui lira les tension/courant d'éventuels panneaux solaires.
Il existe une bibliothèque qui permet de discuter directement avec des INA3221 : « SDL_Arduino_INA3221.h ».
Les variables où on stockera les données acquises pour les courants et tensions batteries sont : « Bat1V, Bat1I, Bat2V, Bat2I, Bat3V et Bat3I ».
On configure le module par « SDL_Arduino_INA3221 Ibat(Ibat_addr, Ibat_shunt_value);) où Ibat_addr est l'adresse du module (ici Ox40) et Ibat_shunt_value la valeur du shunt (commun aux 3 voies). Comme je n'ai pas trouvé la possibilité de configurer une valeur différente de shunt par voie et que j'en avait besoin, j'ai créé d'autres variables IbatX_shunt_value (X allant de 1 à 3) que j'utilise plus loin pour faire la calcul du courant (en fait « Ibat_shunt_value » n'est pas utilisé).
Il faut ensuite démarrer le process : « Ibat.begin(); ».
Pour venir lire les données du INA3221, il faut balayer les trois canaux en récupérant les valeurs de la tension et de la différence de potentiel aux bornes des shunts (« shuntvoltageX » avec X de 1 à 3). On utilise le morceau de programme suivant (en le répétant pour X de 1 à 3) :

```
BatXV = Ibat.getBusVoltage_V(BATX);
shuntvoltageX = Ibat.getShuntVoltage_mV(BATX);
```

Pour avoir le courant, il suffit de diviser, en le corrigeant d'un offset de voltage (ici de 0,16mV), la tension du shunt avec sa résistance :

```
BatXI = (shuntvoltageX - 0.16)/IbatX_shunt_value;
```

L'étalonnage du système se fait en déterminant par une méthode métrologique (définie dans le tutoriel de mise en œuvre), les valeurs exactes de la résistance de shunt et l'offset de mesure.

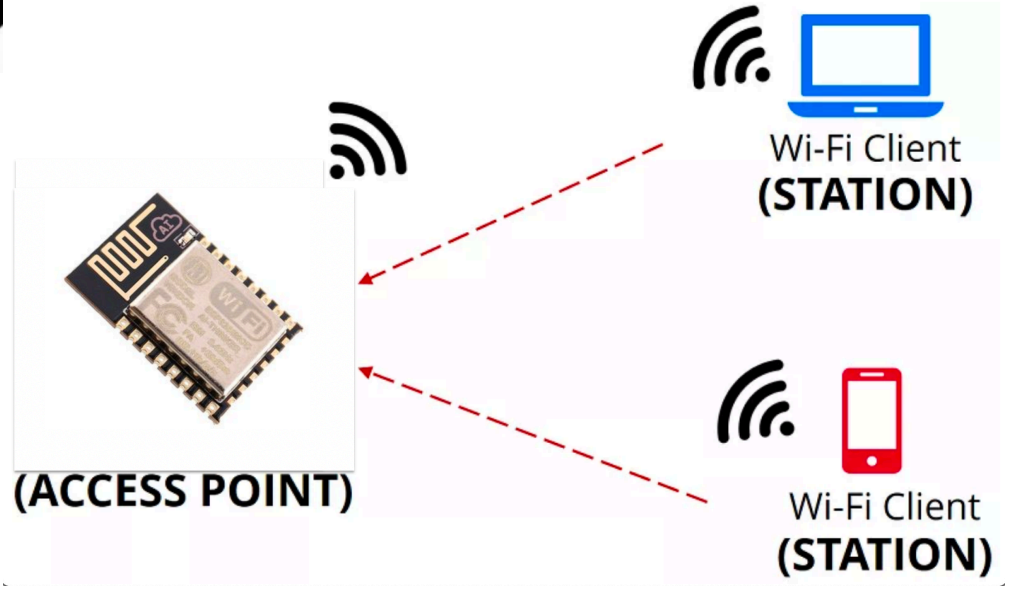
Périodicité des mesures

Les instructions utilisées pour acquérir les données des modules DHT11 et INA3221 sont mises dans une procédure qui sera appelée quand le système devra faire des mesures, c'est : « callACQ() » qui est définie dans un fichier indépendant « FonctionsACQ.h ».
Cette fonction est appelée dans le programme principal toutes les 300ms en utilisant une fonction timer qu'on trouve dans la bibliothèque « Timer.h ». Le timer s'appelle t (« Timer t; »), on le définit dans la boucle setup (« t.every(300, callACQ); ») et on l'active dans la boucle loop (« t.update(); ».

Donc, ici, les données de tension, courant, température et humidité sont rafraichies 3 fois par seconde environ.

Une fois acquises, les données sont toutes mises sous forme d'une ligne au format texte :
parametresbat = String(Temperaturebat,2) + "," + String(RHbat) + "," + String(Bat1V,3) + "," + String(Bat1I,3) + "," + String(Bat2V,3) + "," + String(Bat2I,3) + "," + String(Bat3V,3) + "," + String(Bat3I,3);
C'est ce texte (« parametresbat ») qui sera transféré en wifi pour être affiché sur le terminal web (après retransformation du texte en nombres). On verra plus loin comment c'est transmis et décodé.

Maintenant qu'on a construit le process d'acquisition, il faut construire le WiFi et les protocoles associés.
Tout d'abord définir le type, ici « AccessPoint » c'est à dire pouvoir fournir des données à des stations (les terminaux du type téléphones ou tablettes) qui viendraient s'y connecter.



Toutes les fonctions wifi que nous utiliserons dans ce projet sont réparties dans la bibliothèque « ESP8266WebServer.h »

Mode « ACCESS POINT » :
WiFi.mode(WIFI_AP);

Nom du site (SSID) et son mot de passe pour l'accès :
WiFi.softAP("MonitoringBat", "ConnectBat"); // (SSID, password);

Gestion du serveur

Tant qu'aucun appareil (STATIONS) ne vient se connecter à l'ESP8266, le système tourne en boucle avec environ 3 acquisitions de données batteries et température par seconde sans autres effets.
Si une station se connecte sur le SSID de l'ESP8266 et que son navigateur fait une requête sur l'adresse locale http://192.168.4.1, il faut que l'ESP8266 réponde en actant comme un serveur web et fournisse les fichiers web à la station.
Pour ça, on dispose des fichiers web qui sont stockés dans la partie SPIFFS de la mémoire flash (initialisé par « SPIFFS.begin(); » de la bibliothèque « FS.h ».

Le serveur web est initialisé par les instructions :

```
ESP8266WebServer server(80);
…
server.on("/", HTTP_GET, []() {handleFileRead("/");});
server.onNotFound([]() if (!handleFileRead(server.uri())) server.send(404, "text/plain", "FileNotFound");});
server.begin();
```

Et lancé par l'instruction, l'écoute de stations dans la boucle loop :

```
server.handleClient();
```

Réponse à une station

En réponse à une station qui fait une requête http sur l'ESP8266, on crée un process qui va venir lire les fichiers de l'espace SPIFSS pour les envoyer vers la station via le WiFi. Cette procédure s'appelle « handleFileRead() » et se trouve dans le fichier « FonctionsWifi.h ». Cette procédure appelle une autre procédure qui reconnaît les types de fichiers qui peuvent être présents dans l'espace SPIFSS (text/html, text/css, application/javascript, image/png, image/gif, image/jpeg, image/x-icon, text/xml, application/x-pdf, application/x-zip, application/x-gzip, image/svg+xml).

Il faudra que soit présent dans les scripts web des fichiers de l'espace SPIFFS un process de création d'un websocket qui permettra de créer un canal spécifique pour transférer au site web créé les données acquises en temps réel. Ces données, comme je le disais plus haut sont toutes mises au format texte dans la variable « parametresbat ». Je montrerai plus loin comment on crée ce process.

En réponse au process qui crée le websocket au niveau de la station par le port 81, il faut aussi créer l'autre partie au niveau de l'ESP8266. Pour cela, il existe une bibliothèque spécifique « WebSocketsServer.h » où nous irons chercher les instructions de configuration « WebSocketsServer webSocket(81); », « webSocket.begin(); » et «webSocket.onEvent();» (dans le fichier « FonctionsWifi.h »).
On crée ensuite la procédure qui vient décrire les actions à mener pour faire vivre ce canal websocket.
Il suffit ensuite, à chaque fois que la boucle loop est executée, d'envoyer nos données « parametresbat », à travers le websocket auparavant défini, par l'instruction « webSocket.sendTXT(socketNumber,parametresbat); ».

Pages web du serveur

Ll'ESP8266 a été configuré pour se connecter en WiFi à des stations et il peut, à travers un websocket, transférer les données acquises à la station en temps réel. Il faut maintenant créer les fichiers qui vont constituer les pages web qui seront envoyés à la station et qui seront exécutés par celle-ci.

J'ai utilisé ici le format HTML qui est assez simple à mettre en œuvre associé à des fichiers de description de style au format CSS et surtout le javascript qui m'a permit d'y inclure les beaux objets vu-mètres et courbes proposés par le site https://www.highcharts.com/.
Ces objets javascripts sont assez bien réalisés et ont beaucoup d'options pour les personnaliser (couleurs, formes, texte, …). De plus, ils sont relativement faciles à manipuler pour moi qui ne connais pas grand chose en informatique.

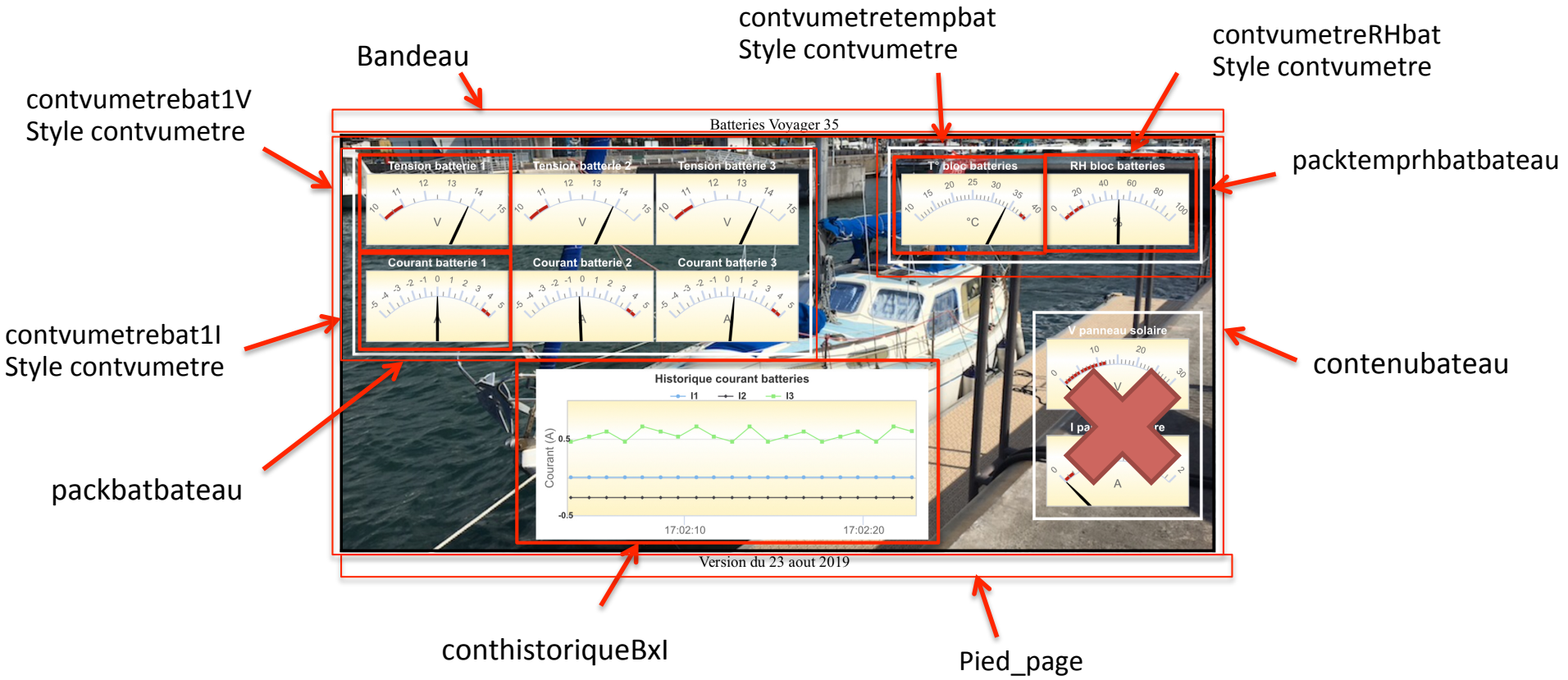
Les pages web que j'ai proposées ici peuvent donc complètement être revues par des pro du web dès l'instant qu'elles respectent la taille totale des fichiers qui doit être inférieure à 3Mo et qu'il y ait la partie qui gère le websocket au port 81 pour récupérer les données acquises en temps réel.

Les pages web que j'ai définies ici sont donc organisées sous cette forme :

- index.html -> c'est la page HTML qui est chargée en premier par la station (voir description plus loin) ;
- Style.css -> c'est là où sont stockés tous les styles de la page HTML (voir description plus loin) ;
- highcharts-more.js.gz et highcharts.js.gz -> sont les versions compressées (.gz) des fichiers des bibliothèques java de Highcharts. Ils contiennent notamment les vu-mètres et la courbe qui défile ;
- jquery-3.1.1.slim.min.js.gz -> est la version compressée (.gz) des bibliothèques javascript nécessaires pour faire fonctionner les objets de Highcharts ;
- apple-touch-icon.png -> c'est l'icône qui apparaît sur le navigateur sur l'onglet de la page. C'est aussi l'icône qui apparaît losqu'on crée une icône sur le bureau d'une tablette ou d'un iphone Apple ;
- Imagefond.jpg -> c'est l'image de fond de la page web (utilisation décrite avec index.html)

Pages web du serveur

Ce fichier est au format HTML et comprend une partie écrite en javascript. La partie javascript est facilement repérable puisqu'elle est entourée des balises <script type = "text/javascript"> et </script>.
Si vous voulez éditer et modifier ce fichier, je vous suggère d'utiliser l'éditeur gratuit disponible sur internet : « Sublime Text ». Cet éditeur reconnaît les différentes balises et il est bien plus aisé de retrouver tous les objets.
Je ne vais pas faire un cour en HTML mais nous retrouverons facilement dans cette partie les éléments constituants de la page, leurs styles CSS et la description javascript pour les vu-mètres et la courbe défilante (ne pas tenir compte des deux cadrans en bas à droite, c'est un ajout que j'ai fait après pour le panneau solaire) :



Objets javascript

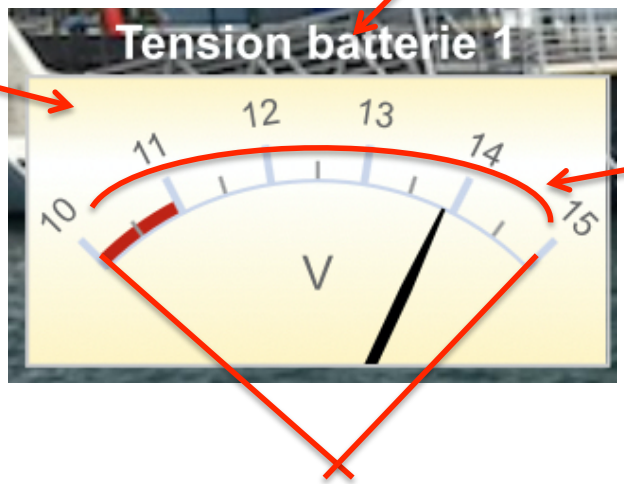
Je ne vais pas décrire ici toutes les options des objets javascript qui sont utilisés mais simplement vous montrer comment on reconnaît les éléments pour éventuellement les modifier. Il y a énormément d'options disponibles sur le site highcharts.com, ceux qui sont un peu curieux s'amuseront follement avec ça. Je vais donc commencer par l'objet vu-mètre dans sa version voltmètre. Vous comprendrez aisément comment sont crées les vumètres courant, température et humidité si vous avez compris le vumètre tension.
Donc, je ne décrirai que l'objet vumètre pour la tension de la batterie 1 et l'objet courbe défilante dans les pages qui suivent.

Objet vu-mètre

```
vmbat1V = new Highcharts.chart('contvumetrebat1V', { ...
});
```

```
chart: {
    type: 'gauge',
    spacing:[2,2,2,2],
    plotBorderWidth: 1,
    backgroundColor: null,
    plotBackgroundColor: {
        linearGradient: { x1: 0, y1: 0, x2: 0, y2: 1 },
        stops: [
            [0, '#FFF4C6'],
            [0.3, '#FFFFFF'],
            [1, '#FFF4C6']
        ]
    },
    plotBackgroundImage: null,
    height: 90
},
```

```
title: {
    text: 'Tension batterie 1',
    style: {
        color: 'white',
        fontSize: "12px",
        fontWeight: 'bold'
    },
    margin: 0,
    widthAdjust: 0
},
```




```
yAxis: [{
    min: 10,
    max: 15,
    minorTickPosition: 'outside',
    tickPosition: 'outside',
    tickInterval: 1,
    minorTickInterval: 0.5,
    minorTickLength: 5,
        labels: {
            rotation: 'auto',
            distance: 15,
            step: 1,
            style: {fontSize: "9px"}
        },
    plotBands: [{
        from: 0,
        to: 11,
        color: '#C02316',
        innerRadius: '100%',
        outerRadius: '105%'
    }],
    style:{"fontSize": "6px"},
    pane: 0,
    title: {
        text: 'V',
        y: -10
    }
}],
```

```
pane: [{
    startAngle: -45,
    endAngle: 45,
    background: null,
    center: ['50%', '140%'],
    size: 150
}],
```

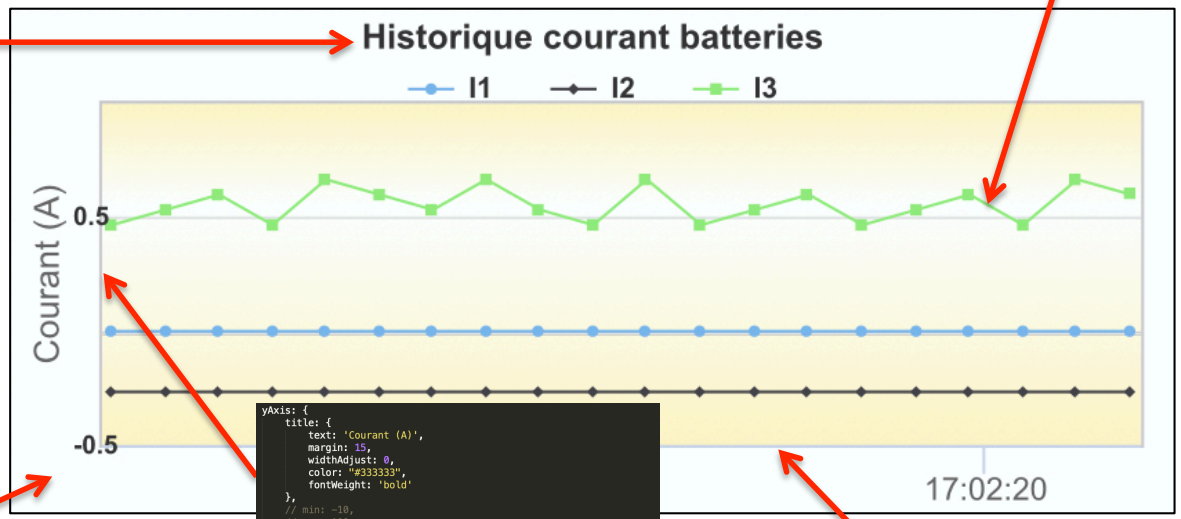
On actualise la valeur de tension affichée sur cet objet par l'instruction javascript suivante :
vmbat1V.series[0].setData([B1Vr]);
Où vmbat1V est la référence de cet objet et B1Vr la variable qui provient du websocket (je montrerai plus loin comment on gère ce websocket en java)

Objets graphe déroulant

```
histBxI = new Highcharts.chart('conthistoriqueBxI', { ...
});
```

```
plotOptions: {
    series: {
        lineWidth: 1,
        enableMouseTracking: false,
        marker: {
            radius: 2
        }
    }
},
```

```
title: {
    text: 'Historique courant batteries',
    style: {
        fontSize: "12px",
        fontWeight: 'bold'
    },
    margin: 0,
    widthAdjust: 0
},
```




```
chart: {
    type: 'line',
    spacing:[2,2,2,2],
    plotBorderWidth: 1,
    backgroundColor: null,
    plotBackgroundColor: {
    linearGradient: { x1: 0, y1: 0, x2: 0, y2: 1 },
    stops: [
        [0, '#FFF4C6'],
        [0.3, '#FFFFFF'],
        [1, '#FFF4C6']
    ]
    },
    animation: Highcharts.svg, // don't animate in old IE
    marginRight: 10,
    marginLeft: 30,
    marginTop: 30,
    lineWidth: 1,
},
legend: {
    align: 'center',
    verticalAlign: 'top',
    layout: 'horizontal',
    y: 10,
    itemStyle: {
        "fontSize": "10px",
        fontWeight: 'bold'
    }
},
```

```
yAxis: {
    title: {
        text: 'Courant (A)',
        margin: 15,
        widthAdjust: 0,
        color: "#333333",
        fontWeight: 'bold'
    },
    // min: -10,
    // max: 100,
    //tickInterval: 10,
    //minorTickInterval: 5,
    minorTickLength: 5,
    tickLength: 10,
    labels: {
        align: 'left',
        x: -10,
        y: 3,
        step: 2,
        style: {color: "#333333",fontSize: "9px",fontWeight: 'bold'}
    },
    plotLines: [{
        value: 0,
        width: 1,
        color: '#808080'
    }]
},
```

Pas de définition de min et max -> échelle auto

```
xAxis: {
    type: 'datetime',
    tickPixelInterval: 150
},
```

On actualise toutes les 1000 ms (1s) la valeur des 3 courants affichés sur cet objet par les instructions javascript suivantes :

```
setInterval(function () {
        var x = (new Date()).getTime(); // current time
        histBxI.series[0].addPoint([x, B1I],true,true,false);
        histBxI.series[1].addPoint([x, B2I],true,true,false);
        histBxI.series[2].addPoint([x, B3I],true,true,false);
}, 1000);
```

Où histBxI est la référence de cet objet et B1I, B2I et B3I les variables qui proviennent du websocket.
x est l'heure qui est affiché sur l'axe des abscisses.

Gestion du websocket côté station

Nous avons vu plus haut comment ouvrir le canal websocket sur le port 81 côté ESP8266. Nous allons voir comment, maintenant, faire la même chose côté station.
Les instructions sont écrites en java.
Tout d'abord la création :

```
var connection = new WebSocket('ws://' + location.hostname + ':81/');
connection.onopen = function () {
    connection.send('Connect ' + new Date());
};
connection.onerror = function (error) {
};
```

Puis maintenant la récupération du message texte qui vient de l'ESP8266 qui provoque un événement quand il arrive à la station :

```
connection.onmessage = function(evt){
    var msgArray = evt.data.split(",");

    B1V = parseFloat(msgArray[2]);
    B2V = parseFloat(msgArray[4]);
    B3V = parseFloat(msgArray[6]);
    B1I = parseFloat(msgArray[3])/1000;
    B2I = parseFloat(msgArray[5])/1000;
    B3I = parseFloat(msgArray[7])/1000;
    tempbat = parseFloat(msgArray[0]);
    rhbat = parseFloat(msgArray[1]);

    if (B1V < 10) {B1Vr = 9.5;} else if (B1V > 16) {B1Vr = 16.5;} else {B1Vr = B1V;};
    if (B2V < 10) {B2Vr = 9.5;} else if (B2V > 16) {B2Vr = 16.5;} else {B2Vr = B2V;};
    if (B3V < 10) {B3Vr = 9.5;} else if (B3V > 16) {B3Vr = 16.5;} else {B3Vr = B3V;};

    if (B1I < -5) {B1Ir = -5.5;} else if (B1I > 5) {B1Ir = 5.5;} else {B1Ir = B1I;};
    if (B2I < -5) {B2Ir = -5.5;} else if (B2I > 5) {B2Ir = 5.5;} else {B2Ir = B2I;};
    if (B3I < -5) {B3Ir = -5.5;} else if (B3I > 5) {B3Ir = 5.5;} else {B3Ir = B3I;};

    if (tempbat < 10) {tempbatr = 7;} else if (tempbat > 40) {tempbatr = 43;} else {tempbatr =
        tempbat;};

    vmbat1V.series[0].setData([B1Vr]);
    vmbat2V.series[0].setData([B2Vr]);
    vmbat3V.series[0].setData([B3Vr]);
    vmbat1I.series[0].setData([B1Ir]);
    vmbat2I.series[0].setData([B2Ir]);
    vmbat3I.series[0].setData([B3Ir]);
    vmtempbat.series[0].setData([tempbatr]);
    vmrhbat.series[0].setData([rhbat]);
};
```

On y récupère la variable « evt.data » qui contient le contenu de ce que l'ESP8266 a envoyé et qui contient tous les paramètres mesurés sous forme texte (variable « parametresbat » côté ESP8266). Chaque élément de mesure est séparé par une virgule.
Une fonction javascript très pratique, « split », permet de récupérer chaque élément sous la forme d'un tableau.
Il suffit ensuite de transformer le texte en nombre flottant (« parsefloat ») et on est prêt à les afficher sur nos vu-mètres et graphes.
Un petit traitement avant affichage permet de cadrer les valeurs entre 9,5V et 16,5V pour les vu-mètres tensions et entre -5,5A et +5,5A pour les vu-mètres courant (sinon l'aiguille disparaît, c'est moche)