

RAPPORT DE PROJET DE RECHERCHE TUTEUR

4^{ème} année Automatique, Electronique et Systèmes Embarqués

Vers une maison intelligente...



Sophie ROUGEAUX
Aurélien LEMAITRE

Baptiste DELGA
Jonathan MALIQUE

Tuteur : **Thierry MONTEIL**

Juin 2018

SOMMAIRE

INTRODUCTION	1
I. Architecture globale du système.....	2
A. Schéma du système	2
B. Mise en œuvre.....	2
II. Organisation et méthodologie	4
A. Partage des tâches, différentes parties et outils choisis	4
B. Planning et suivi du projet.....	4
III. OM2M et les requêtes HTTP, points centraux du projet	5
A. OM2M, plateforme d'interconnexion	5
B. Les requêtes HTTP	5
IV. Capteurs, actionneurs : l'ESP et la carte Intel Edison	8
A. L'ESP et le servomoteur.....	8
B. L'Edison.....	9
V. Reconnaissance faciale et Visual Studio.....	12
A. Prise en main du matériel, du logiciel et premiers essais	12
B. Réalisation finale : détection d'expressions	12
VI. L'application et Android Studio	14
A. Prise en main du logiciel	14
B. Idée de base de l'application.....	14
C. Mise en pratique	14
CONCLUSION	16

INTRODUCTION

Le sujet de notre projet s'inscrit dans l'évolution technologique actuelle des maisons, de plus en plus connectées et intelligentes. Mettant en œuvre la reconnaissance faciale et l'Internet des Objets, il a pour but de simplifier et personnaliser des actions courantes à la maison.

Objectifs initiaux :

Nous souhaitons implémenter un processus de reconnaissance faciale à l'entrée de la maison. Selon la personne détectée, un scénario sera mis en place. Par exemple, faire couler un café pour la personne X, ou allumer la lumière dans une certaine pièce pour la personne Y. Les consignes et les visages seront référencés via une application.

À la suite de notre état de l'art, nous avons conclu qu'OM2M conçue au LAAS, à Toulouse, constituerait notre plateforme d'interconnexion pour nos objets connectés. Nous avons étudié les différents protocoles de communication entre objets connectés que nous pouvons choisir, et nous avons également vu qu'il était possible de faire de la reconnaissance faciale avec une caméra et des capteurs infrarouges.

Dans ce rapport, nous présenterons tout d'abord le système global que nous avons créé avec nos choix de réalisation, puis nous expliquerons le partage des tâches et le planning du projet.

Notre projet étant décomposable en quatre sous-systèmes, les chapitres III, IV, V, VI détailleront la mise en place de chaque sous-système avec les éventuelles difficultés rencontrées. Nous concluons sur les résultats obtenus, les perspectives du projet et sur ce qu'il nous a apporté.

I. Architecture globale du système

A. Schéma du système

Notre système est décomposable en trois grandes parties : le contrôle des capteurs et actionneurs, l'interface Homme-Machine, et la plateforme d'interconnexion, représentées ci-dessous.

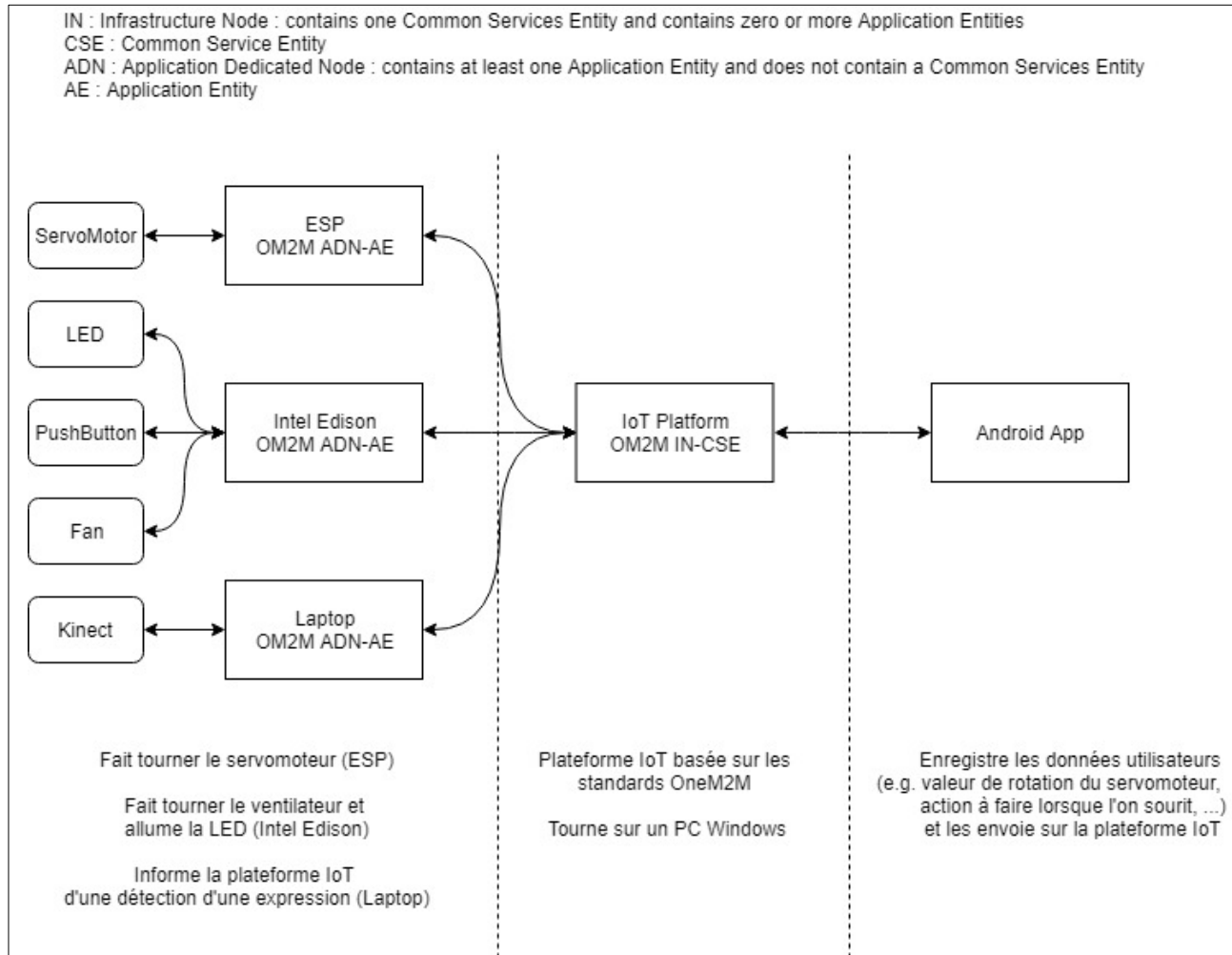


Figure 1 : Architecture de notre système

Note : La détection de personne a été remplacée par de la détection d'expression du visage. Ce choix est expliqué dans le chapitre V.

B. Mise en œuvre

Un scénario d'utilisation de ce projet possible est le suivant :

1. Le servomoteur est placé à un endroit souhaité de l'utilisateur. Pour l'exemple, il sera placé au niveau de la cafetière.
2. L'utilisateur rentre chez lui et passe devant la Kinect. S'il ouvre la bouche, le servomoteur s'active et son café coule. S'il fronce les sourcils, le ventilateur s'allume. S'il sourit, une LED s'allume pendant 2 secondes.
3. L'utilisateur peut modifier sur l'application Android ce qu'il souhaite voir se passer quand il ouvre la bouche, fronce les sourcils ou sourit.

Nous utilisons :

- Un servomoteur commandé par un ESP, et deux engrenages pour qu'il réalise des mouvements latéraux, cela lui permet d'appuyer ou de tirer.
- Un bouton poussoir, une LED et un ventilateur branchés à une carte Intel Edison.
- Une Kinect, possédant capteurs infrarouge et caméra RGB, est branchée à un PC sur lequel VisualStudio s'exécute.
- Une application est installée sur une tablette sous Android.
- La plateforme OM2M qui permet la communication entre ces machines s'exécute sur le PC.

Tous sont connectés au même réseau Wifi.

II. Organisation et méthodologie

A. Partage des tâches, différentes parties et outils choisis

Nous avons travaillé en parallèle sur les différentes parties du projet.

- Jonathan a travaillé sur l'Interface Homme-Machine. Il a codé une application Android qui permet de modifier les préférences d'une personne, ses désirs quand elle est chez elle. Pour cela, l'application envoie des requêtes HTTP à OM2M de manière à ce que l'ESP et la carte Edison puissent lire les nouvelles tâches à effectuer. Pour le développement de l'application, nous avons utilisé une tablette Android avec le logiciel de conception Android Studio (code en Java et xml) et les requêtes http.

- Baptiste a travaillé sur la reconnaissance faciale à l'aide de la Kinect. Il a complété un code présent dans le Software Development Kit (SDK) de la Kinect pour permettre de reconnaître différentes expressions de visage qui seront transmises à OM2M et correspondront à différentes actions dans la maison. Pour cette partie, deux caméras étaient disponibles : Intel Real Sense et la Kinect. Pour la réalisation du code, il s'agissait du logiciel de développement Visual Studio avec la possibilité de coder en C# ou C++.

- Aurélien a travaillé d'abord sur OM2M, son fonctionnement, la compréhension des différentes requêtes. Il a ensuite rejoint Baptiste pour travailler sur la reconnaissance faciale, notamment sur les envois de requêtes HTTP lorsqu'une expression d'un visage a été reconnue.

- Sophie a travaillé sur les capteurs et actionneurs du système (autres que la Kinect). Elle a notamment programmé le contrôle du servomoteur sur l'ESP, la gestion des différents capteurs et actionneurs sur l'Intel Edison. Elle a travaillé sur les requêtes HTTP à envoyer et sur les librairies à utiliser pour l'HTTP. Les langages C et C++ ont été utilisés pour gérer les capteurs et actionneurs.

Finalement, nous avons tous pris en main OM2M et les requêtes http puisque c'est ce qui connecte nos capteurs, actionneurs et application.

B. Planning et suivi du projet

Durant tout le projet, nous sommes restés avec la même répartition des tâches excepté Aurélien qui a travaillé avec Baptiste sur la Kinect vers la fin du projet. Nous nous retrouvions régulièrement pendant les créneaux de Projet Tuteuré, afin de travailler et de faire un point sur l'avancée de chacun d'entre nous dans sa partie.

Un mois avant la fin du projet nous avons pu réaliser une réunion de rencontre entre les concepteurs de la plateforme OM2M et notre groupe. Cela nous a permis de répondre à plusieurs questions concernant la plateforme ainsi qu'à améliorer l'architecture de notre projet (simplification en enlevant la Raspberry Pi qui aurait servi de gateway, finalement peu pertinent pour le projet), nous les remercions donc beaucoup pour leur participation.

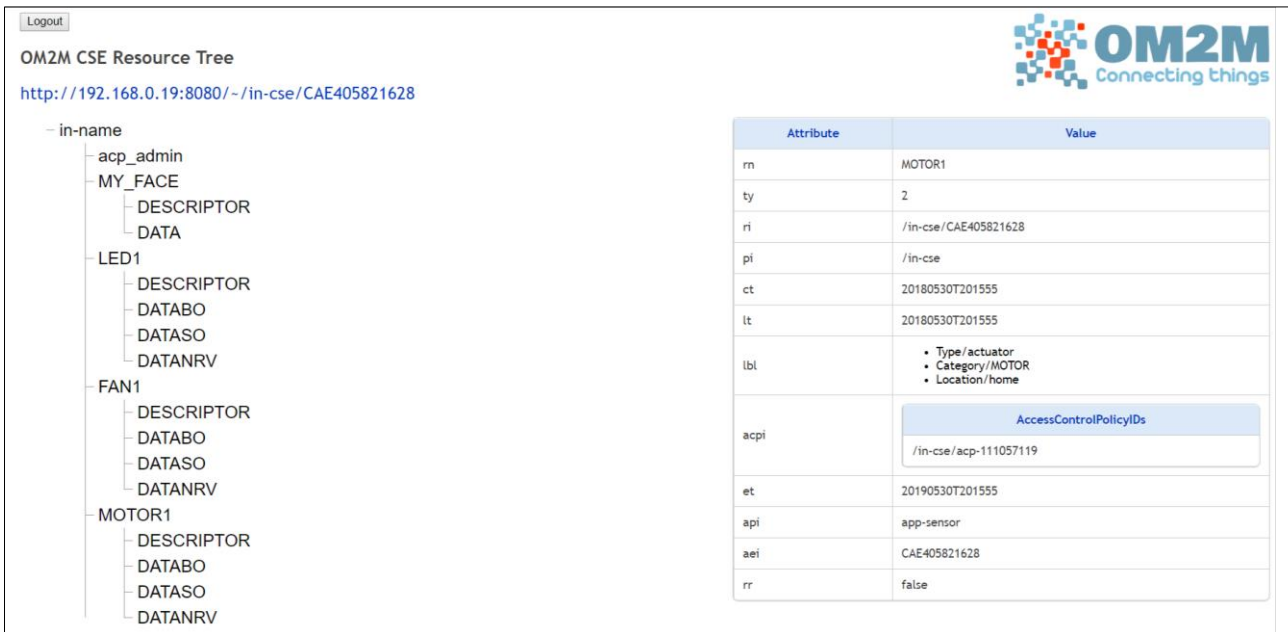
Vers la fin du projet, nous nous sommes réservés de nombreuses heures de travail en commun afin de relier les parties du projet entre elles et de tester les nombreuses connexions entre nos différents systèmes.

III. OM2M et les requêtes HTTP, points centraux du projet

A. OM2M, plateforme d'interconnexion

Nous allons parler de cette plateforme très souvent dans la suite de ce rapport car son but principal est d'interconnecter les différentes parties qui composent notre projet.

OM2M a pour objectif de faciliter la liaison entre les différents capteurs et actionneurs connectés. En effet, il permet de regrouper ensemble les appareils utilisant le même protocole (WiFi, Ethernet, ...) pour pouvoir communiquer avec des appareils utilisant un autre protocole. OM2M est donc composé d'une interface WEB où l'on peut accéder à la donnée de chaque capteur et actionneur. L'image suivante nous montre l'interface WEB OM2M de notre projet :



The screenshot displays the OM2M CSE Resource Tree interface. On the left, a tree structure shows the hierarchy of resources under the 'in-name' node. The tree includes nodes for 'acp_admin', 'MY_FACE', 'LED1', 'FAN1', and 'MOTOR1', each with sub-nodes for 'DESCRIPTOR', 'DATABO', 'DATASO', and 'DATANRV'. On the right, a table lists attributes and their values for the selected resource.

Attribute	Value
rn	MOTOR1
ty	2
ri	/in-cse/CAE405821628
pi	/in-cse
ct	20180530T201555
lt	20180530T201555
lbl	<ul style="list-style-type: none">Type/actuatorCategory/MOTORLocation/home
acpi	<div>AccessControlPolicyIDs</div> <div>/in-cse/acp-111057119</div>
et	20190530T201555
api	app-sensor
aei	CAE405821628
rr	false

Figure 2 : Interface WEB OM2M

On peut remarquer ainsi que nous avons 3 actionneurs (une LED, un Ventilateur et un Moteur) et un capteur (la Kinect). Ainsi, chaque capteur ou actionneur possède deux dossiers, un dossier DESCRIPTOR qui possède la description du capteur ou de l'actionneur et un dossier DATA, c'est ce dossier le plus important. Le dossier DATA sera composé des 10 dernières valeurs. Pour notre projet, il y a plusieurs dossiers DATA car nous avons voulu séparer les dernières valeurs récupérées en fonction des expressions du visage effectuées par l'utilisateur.

Enfin, on peut remarquer que la structure de notre projet est relativement simple (pas de gateway), nous aurions pu utiliser un simple server web plutôt qu'OM2M. Cependant, pour l'intérêt pédagogique de ce projet nous avons choisi d'utiliser cette plateforme IoT.

B. Les requêtes HTTP

OM2M fonctionne grâce aux requêtes HTTP. Il existe plusieurs protocoles de requêtes HTTP, mais dans ce projet, nous en utilisons seulement deux : les requêtes GET et POST. Pour connaître le fonctionnement de ces requêtes dans l'environnement OM2M, nous avons utilisé « Eclipse Wiki » consacré

à OM2M¹, une riche base pour apprendre à utiliser la plateforme. Cela nous a permis de comprendre comment créer toutes les instances du IN-CSE de OM2M.

Une requête HTTP est composé de quatre champs types, chacun pouvant avoir des valeurs ou non :

- URL : c'est l'adresse Internet à laquelle nous allons interroger le serveur.
- Method : C'est le protocole de la requête HTTP, cela peut être POST ou GET dans notre cas.
- Header : c'est dans ce champ que nous allons donner tous les en-têtes de la requête. Un en-tête peut par exemple être l'identifiant et le mot de passe pour se connecter au serveur, le type de langage utilisé dans le body.
- Body : C'est le cœur de la requête POST. Il est vide pour la requête GET. Il possédera toutes les informations que nous voulons transmettre au serveur (en xml ou json).

La première requête est la plus basique, c'est la requête GET. Elle permet de récupérer la donnée à l'endroit que l'on a défini (dans notre cas, dans le serveur OM2M). Elle est présentée comme suivant :

Field	Value
URL	http://127.0.0.1:8080/~in-cse
Method	GET
Header	X-M2M-Origin: admin:admin Accept: application/xml
Body	(empty)

Figure 3 : Request GET

Ainsi, cette requête GET va demander la valeur se trouvant à l'adresse <http://127.0.0.1:8080/~in-cse>. Cette requête va alors amener une réponse qui sera de la forme suivante :

Field	Value
Status	200 OK
Body	<pre><?xml version="1.0" encoding="UTF-8"?> <m2m:cb xmlns:m2m="http://www.onem2m.org/xml/protocols" rn="in-name"> <ty>5</ty> <ri>/in-cse</ri> <ct>20151104T150228</ct> <lt>20151104T150228</lt> <acpi>/in-cse/acp-563660429</acpi> <cst>1</cst> <csi>in-cse</csi> <srt>1 2 3 4 5 9 14 15 16 17 23</srt> <poa>http://127.0.0.1:8080</poa> </m2m:cb></pre>

Figure 4 : HTTP response GET

Ainsi, cette réponse aura comme status 200 OK ce qui signifie qu'il n'y a eu aucun problème. La valeur que nous recherchons sera après la section <poa>. Dans cet exemple, la requête GET a été effectuée sur un dossier donc elle ne renvoie aucune donnée.

¹ Eclipse Wiki pour OM2M, requêtes http et création des instances, disponible sur http://wiki.eclipse.org/OM2M/one/REST_API

Pour la méthode POST, c'est le même principe sauf pour le body qui, ici, contient des données que l'on veut transmettre :

Field	Value
URL	http://127.0.0.1:8080/~in-cse
Method	POST
Header	X-M2M-Origin: admin:admin Content-Type: application/xml;ty=2
Body	<pre><m2m:ae xmlns:m2m="http://www.onem2m.org/xml/protocols" rn="MY_SENSOR" > <api>app-sensor</api> <lbl>Type/sensor Category/temperature Location/home</lbl> <rr>false</rr> </m2m:ae></pre>

Figure 5 : Request POST

Ici, on peut reconnaître grâce au body que l'on veut ajouter dans OM2M un dossier MY_SENSOR qui correspond à un capteur de température. La réponse à cette requête est en majeure partie la même que pour une requête GET, la différence sera sur ce qui a été écrit dans le body lors de l'envoi.

Ainsi, grâce à ces deux requêtes, on pourra lire et ajouter des valeurs dans notre serveur OM2M et faire fonctionner tous les capteurs ensembles.

IV. Capteurs, actionneurs : l'ESP et la carte Intel Edison

Pour cette partie du projet, nous avons comme actionneurs :

- Un servomoteur connecté à un ESP8266 (le « Nelson »²).
- Un ventilateur connecté à la carte Intel Edison, programmé en C/C++.
- Une LED connectée à la carte Intel Edison.

Nous avons comme capteur :

- Un bouton poussoir connecté à la carte Intel Edison.

Pour tous ces capteurs et actionneurs, un ou deux ESP auraient suffi mais nous avons la carte Intel Edison à disposition, bien plus puissante, et qui permet également de mettre en œuvre nos connaissances en C/C++.

Initialement, nous souhaitions travailler uniquement avec la carte Intel Edison, mais il était difficile de la prendre en main. En effet, la documentation sur internet est relativement réduite. De plus, cette année, le BE de C++ ayant commencé tard, nous n'avons pas pu nous appuyer dessus pour avancer notre projet tuteuré, bien que cela aurait été très intéressant.

Finalement, l'intérêt de posséder deux cartes est de pouvoir les placer dans deux pièces différentes, ce qui est concrètement très intéressant et proche d'un réel besoin.

A. L'ESP et le servomoteur

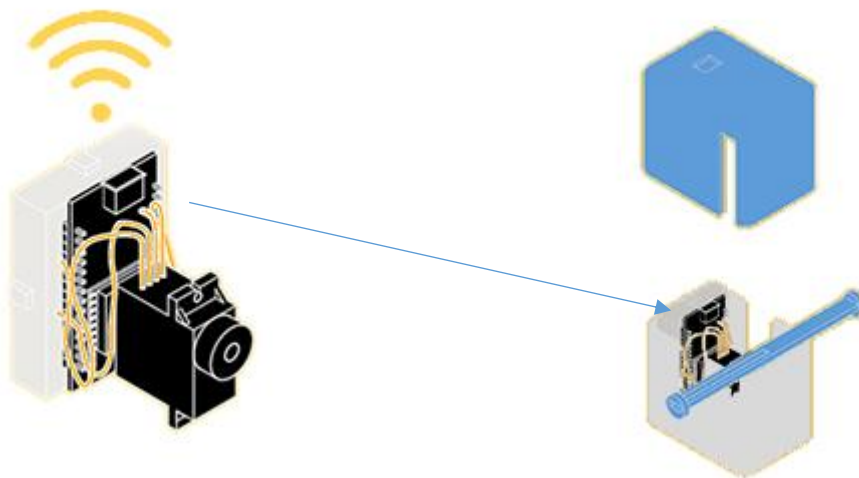


Figure 6 : Le "Nelson" : Un servomoteur qui peut pousser et tirer

L'ESP et le servomoteur sont assemblés dans un boîtier imprimé en 3D. Le servomoteur est associé à deux engrenages, ce qui permet de réaliser des mouvements pour pousser ou tirer. Le servomoteur est actionné lorsqu'une expression est reconnue et que l'utilisateur a indiqué sur l'application Android que cette expression est associée au moteur.

² Créé par Maxime Castelli en 2015, le « Nelson » est un petit module connecté constitué d'un servomoteur et d'une carte ESP. Plus d'informations ici : <http://maximecastel.li/?/projects/Nelson/>

Il réalise donc les fonctions suivantes :

- A son initialisation, s'identifier sur OM2M en tant qu'application MOTOR1, possédant un DESCRIPTOR et trois DATA : DATABO, DATASO, et DATANRV. *Si l'utilisateur a indiqué que le moteur s'allume lorsque l'utilisateur ouvre la bouche, il sera écrit « data : 1 » dans le dernier Content Instance de DATABO. S'il doit s'activer lorsque l'utilisateur sourit, ce sera écrit dans DATASO, et quand l'utilisateur fronce les sourcils, dans DATANRV.* On initialise toutes les Content Instances à 0.
- Puis, il va régulièrement vérifier la valeur de la dernière expression reconnue (BO, SO ou NRV) dans MY_FACE/DATA.
- Si cette valeur est à BO, il vérifie la dernière Content Instance dans MOTOR1/DATABO, et réalise un mouvement d'avant en arrière (permet d'appuyer physiquement sur un bouton) si celle-ci vaut 1. Si la valeur est SO, il va vérifier dans MOTOR1/DATASO, et si c'est NRV, il va vérifier dans MOTOR1/DATANRV.

Comme la détection du visage était un point sensible du projet, nous avons commencé par simuler les valeurs de MY_FACE/DATA par un bouton poussoir qui écrivait dans MY_FACE/DATA lorsqu'il était appuyé. Tout ce travail n'apparaît pas dans le projet final car ce bouton poussoir n'est plus utile maintenant que la détection de visage fonctionne.

Difficultés rencontrées : Le servomoteur est très simple à contrôler grâce à la librairie Arduino. En revanche, il a été plus fastidieux d'analyser la réponse d'OM2M pour récupérer la valeur de la position du servomoteur. Cette partie du code prend beaucoup de temps et augmente fortement le temps de réponse du servomoteur, l'ESP est peu puissante comparée à l'Edison.

B. L'Edison

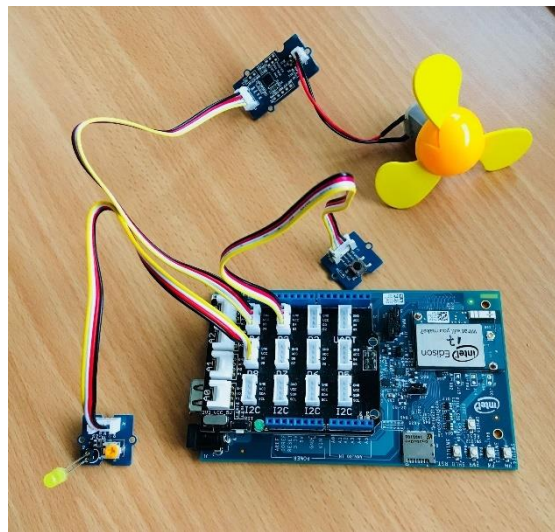


Figure 7 : Montage avec la carte Intel Edison, le ventilateur, le bouton poussoir et la LED

La carte Edison réalise les fonctions suivantes :

- A son initialisation, identifier ses capteurs et actionneurs sur OM2M (LED1, FAN1), ainsi que la Kinect (MY_FACE) car il était plus simple de réaliser l'initialisation sur l'Edison que sur VisualStudio.
- Puis, on va régulièrement vérifier la valeur de la dernière expression reconnue (BO, SO ou NRV) dans MY_FACE/DATA.
- Si cette valeur est à BO, on vérifie la dernière Content Instance dans FAN1/DATABO et dans LED1/DATABO (l'Edison possède deux actionneurs), et on actionne le ventilateur et la LED s'il est

écrit 1. Si la valeur dans MY_FACE/DATA est SO, on va vérifier dans FAN1/DATASO et dans LED1/DATASO. Si c'est NRV, on va vérifier dans .../DATANRV.

- Le bouton poussoir permet d'arrêter manuellement le ventilateur (pas de liaison OM2M).

Pour réaliser les requêtes HTTP, nous utilisons la bibliothèque cURL, très complète et qui permet une grande flexibilité dans la construction des requêtes.

Pour simplifier le code, nous avons créé des fonctions *sendPost(string url, string ty, string body)* (voir ci-dessous) et *sendGet(string url)* qui utilisent cette bibliothèque cURL, et que nous pouvons utiliser en toute simplicité dans le corps du programme.

```
int sendPost(string url, string ty, string body){
    CURL *curl;
    CURLcode res;
    struct curl_slist *list = NULL; //list of headers

    // get a curl handle
    curl = curl_easy_init();

    if(curl) {
        // URL
        curl_easy_setopt(curl, CURLOPT_URL, url.c_str());

        // METHOD
        curl_easy_setopt(curl, CURLOPT_CUSTOMREQUEST, "POST");

        // HEADERS
        list = curl_slist_append(list, "X-M2M-Origin: admin:admin");
        string h = "Content-type: application/xml;ty=";
        h = h + ty;
        list = curl_slist_append(list,h.c_str());
        curl_easy_setopt(curl, CURLOPT_HTTPHEADER, list);

        //DATA
        curl_easy_setopt(curl, CURLOPT_POSTFIELDS, body.c_str());

        // SEND REQUEST
        res = curl_easy_perform(curl);

        curl_slist_free_all(list); // free the list again
    }

    if(res != CURLE_OK){
        cout<<"curl_easy_perform() failed"<<endl;
        return 1;
    }

    curl_easy_cleanup(curl);

    return 0;
}
```

Les url et les corps des requêtes que nous utilisons dans le code sont définis au début du fichier. Ils sont nombreux, cela a été un travail fastidieux mais nécessaire pour un code clair. Un exemple d'url, de corps de requête et d'appel à la fonction sendPost pour poster une Content Instance de valeur « data : 0 » dans FAN1/DATABO est donné ci-dessous.

```
string url = "http://192.168.43.234:8080/~in-cse";

string url_fan1_databo = url + "/in-name/FAN1/DATABO";

string body_fan1_dataCI0 =
"<m2m:cin xmlns:m2m=\"http://www.onem2m.org/xml/protocols\">
<cnf>message</cnf>
<con>
    <obj>
        <str name=\"appId\" val=\"FAN1\"/>
        <str name=\"category\" val=\"fan\"/>
        <int name=\"data\" val=\"0\"/>
        <int name=\"unit\" val=\"0:off,1:on,2:R\"/>
    </obj>
</con>
</m2m:cin>";

sendPost(url_fan1_databo, "4", body_fan1_dataCI0)
```

Ce code a été réalisé en C++. Le corps du programme est constitué d'une boucle infinie qui teste les valeurs reçues d'OM2M et agit en conséquence.

Pour conclure, la carte Edison associée à la connectique Groove est un outil puissant et multifonctionnel. Son plus gros défaut a été le manque de documentation disponible sur internet, et le début tardif du BE de C++ qui aurait permis de prendre plus vite la carte en main et d'utiliser les classes héritées du C++ pour une programmation plus élégante.

V. Reconnaissance faciale et Visual Studio

A. Prise en main du matériel, du logiciel et premiers essais

Pour la partie concernant la reconnaissance faciale, l'objectif était de pouvoir identifier des personnes selon leur visage, pour ensuite, selon la personne identifiée, envoyer la donnée vers OM2M via une requête et ainsi effectuer un scénario de commande sur les actionneurs.

Nous avons le choix entre la caméra Intel Real Sense et la caméra **Kinect** ; nous avons décidé d'utiliser la Kinect sous Visual Studio pour une plus simple utilisation grâce à une documentation plus complète et des codes « open source » plus nombreux mais aussi car nous n'avons jamais été en mesure de faire fonctionner correctement la caméra Intel.

Pour démarrer, il a fallu télécharger toutes les ressources nécessaires : Visual Studio pour la compilation et l'exécution du code, le SDK 2.0 Kinect pour et le Developer Toolkit v1.8.0 avec les outils Kinect Studio et Kinect Explorer ainsi que plusieurs exemples de code en C# ou C++. Sans oublier de télécharger les pilotes nécessaires à la détection de la Kinect sur le PC en faisant attention à prendre la bonne version.

Une des premières difficultés fut de réussir à compiler et exécuter les exemples de code fournis concernant le *Face Tracking* (librairies manquantes, sans doute dû à la version de la Kinect).

Une fois la prise en main de Visual Studio effectuée, nous avons cherché différents codes « open source » permettant de reconnaître un visage et d'identifier la personne. Nous avons rencontré divers problèmes lors de multiples tentatives de génération de ces types de code (en C#) : projet impossible à compiler sous Visual Studio, erreurs à la compilation avec des librairies/bibliothèques manquantes ou de mauvaises versions (disponibles qu'en 1.8 ou inversement qu'en 2.0), problèmes à l'exécution du projet (ne donnant pas le résultat escompté). Après plusieurs tentatives de débogage et par manque de temps, nous avons décidé de modifier l'objectif initial qui était de reconnaître et d'identifier différents visages rentrés dans une base de données.

B. Réalisation finale : détection d'expressions

Nous avons donc choisi de prendre un exemple de code présent dans le SDK comme base pour ensuite le modifier de telle sorte à identifier d'une certaine manière une personne ou une expression sur son visage pour pouvoir envoyer des données à la plateforme OM2M.

Pour ceci, le choix d'un code en C++ fut optimal, étant plus familier que le C#, et ainsi pour faciliter la compréhension du contenu sachant que des changements dans le code sont nécessaires.

Il nous a fallu comprendre les actions de chaque fonction pour trouver les fonctions à modifier pour permettre de détecter n'importe quel visage tout d'abord, puis dans un second temps insérer des types d'expression de visages à identifier. Cela grâce à des variables flottantes indiquant les coordonnées de quelques points du visage. Dans notre cas, les points concernant la bouche, la mâchoire, le front, et les yeux.

En testant différentes expressions de visage sur plusieurs personnes et en observant les valeurs de ces variables que nous affichons sur la console, nous avons pu déterminer quelques expressions du visage que nous arrivons à déceler en fixant des seuils de valeur fonctionnant pour tout visage.

Les trois expressions de visage choisies sont :

- Les froncements de sourcil avec deux variables regroupant les points de données du front. On observe la différence entre le haut du front (en-dessous des cheveux) et le bas du front (au-dessus des sourcils),
- Le sourire avec deux variables donnant les points aux deux coins de la bouche,

- La bouche grande ouverte avec une variable donnant des indications sur la mâchoire.

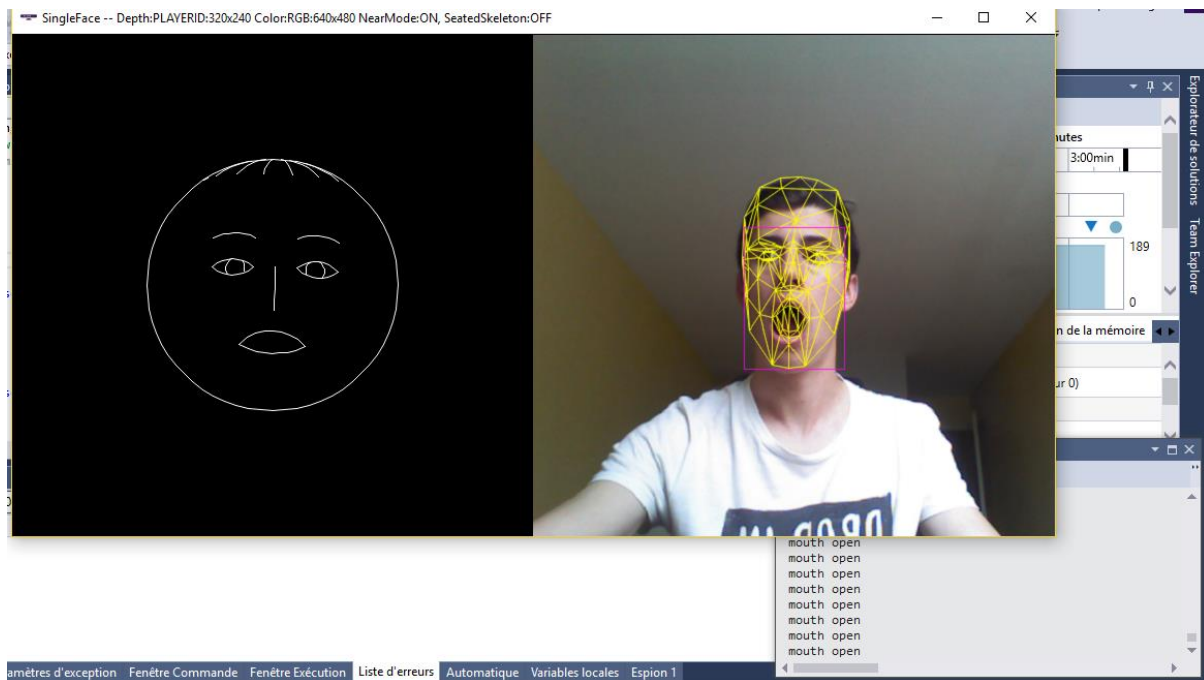


Figure 8 : Exemple de test d'un scénario (ici bouche ouverte)

Selon les valeurs de ces variables, nous pouvons donc identifier l'expression du visage enregistrée par la Kinect, et ainsi envoyer des requêtes vers OM2M après avoir bien vérifié que les bons messages de test s'affichaient et correspondaient bien aux expressions correspondantes comme ci-dessus avec *mouth open* qui s'affiche bien comme espéré.

Lorsqu'une expression d'un visage est découverte, il faut alors le communiquer à OM2M pour que les différents actionneurs agissent. Pour cela, nous avons cherché différentes solutions. En utilisant le logiciel Postman, nous avons pu traduire la requête HTTP en C++ avec la librairie cURL. Cette librairie a été créée spécifiquement pour envoyer des requêtes à une URL spécifiée par nous-même.

Lors de la mise en place de cette librairie, nous avons eu de nombreux problèmes notamment dus à Visual Studio puisque ce logiciel devait connaître l'emplacement exact de cette librairie, mais aussi ajouter les différentes DLL (bibliothèques logicielles) dans le système de l'ordinateur. Néanmoins après la correction de ces erreurs assez compliquée, les requêtes étaient opérationnelles.

VI. L'application et Android Studio

A. Prise en main du logiciel

La première difficulté concernant cette partie de la conception était bien entendu de prendre en main le logiciel de développement d'une application sous Android. Pour cela, nous avons utilisé les cours en ligne d'Openclassrooms³ qui sont très bien faits, puis commencé à coder notre application.

Les applications sont divisées en activités. Chaque activité correspond à une page/menu de l'application. Android Studio permet de créer une activité graphiquement et visuellement ce qui est plus facile pour commencer. Chaque layout (design d'une activité) est écrit en xml et modifiable à volonté.

Ensuite, il faut coder en java les différentes actions applicables à chaque activité. Pour cela, par exemple, on inclut une fonction à effectuer dans le xml d'un bouton (onClick) que l'on va coder en Java et qui sera appelée à chaque clic sur le bouton en question.

B. Idée de base de l'application

Notre idée de base de l'application est de seulement permettre une mise à jour des actions souhaitées. C'est-à-dire qu'à la mise en place du système dans la maison, on règle les paramètres souhaités quand on est détecté par la caméra dans la tablette, puis on clique sur un bouton de mise à jour qui envoie tous ces paramètres à OM2M. Ce sont ensuite l'ESP et la carte Edison qui s'occupent d'aller retrouver ces valeurs dans OM2M lors de la détection d'une expression du visage. On peut bien entendu changer les actions souhaitées quand on le veut.

Au départ, on pensait pouvoir enregistrer son visage dans un autre menu et en appuyant sur un bouton d'enregistrement (en se plaçant face à la caméra). Finalement, nous n'avons pas pu effectuer une différenciation des visages et nous différencions donc des expressions du visage.

C. Mise en pratique

Nous avons donc tout d'abord commencé par la création des activités graphiquement seulement, sans donner de fonctions aux boutons. Voici les captures des 4 différentes activités disponibles :

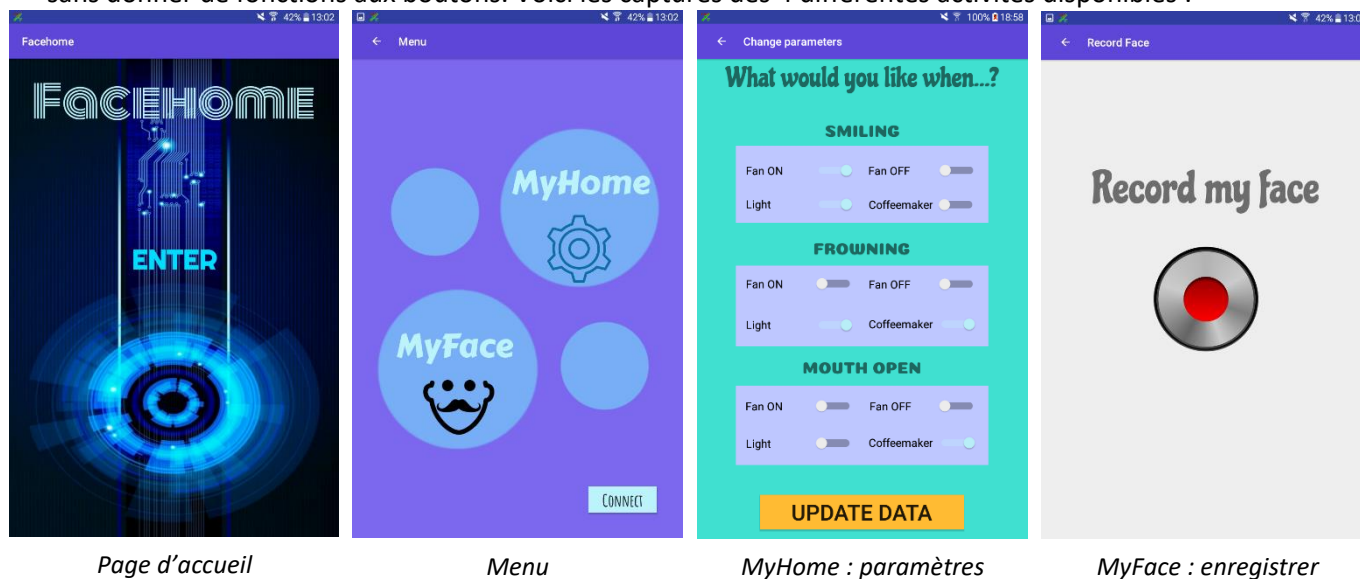


Figure 9 : Captures d'écran de l'application

³ « Créez des applications pour Android » disponible sur <https://openclassrooms.com/courses/creez-des-applications-pour-android>.

On a donc une première activité qui correspond à la page d'accueil avec un clic nécessaire pour rentrer dans le Menu. Le Menu comporte deux boutons, sachant que le bouton « MyFace » permet de rentrer dans l'activité d'enregistrement de visage qui n'a finalement pas été fait.

La partie la plus importante concerne donc l'activité « MyHome » avec le bouton « Update Data » qui permet de mettre à jour les valeurs dans OM2M. Pour cela, il a fallu coder des requêtes HTTP en Java. Nous avons utilisé la librairie java OkHttp développée par Square. Voici l'exemple simplifié de l'une de nos requêtes POST :

```
Request request = new Request.Builder()
    .url(urlom)
    .header("X-M2M-Origin", "admin:admin")
    .addHeader("Content-Type", "application/xml;ty=4")
    .post(body)
    .build();

client.newCall(request).enqueue(this);
```

Ici, on construit la requête avec un URL destinataire « urlom » défini auparavant, avec deux Header et un corps « body » qui a été défini en XML auparavant également. Le « .build » permet de finir de construire cette requête et la dernière ligne de l'envoyer au serveur.

Ensuite il faut définir 3 fonctions de vérification : « isConnected » qui permet de prévenir quand il y a un problème de connexion au moment de l'envoi, « onFailure » qui permet de prévenir en cas d'échec de la requête, et « onResponse » qui permet d'afficher la réponse du serveur même si celle-ci correspond à une erreur.

Dans l'activité « MyHome », il y a trois parties qui correspondent à trois expressions du visage, dans l'ordre : un sourire, un froncement des sourcils, et l'ouverture de la bouche. Dans chaque partie on peut décider des actions à effectuer selon l'expression. Une fois le choix fait, il suffit de cliquer sur le bouton « Update Data » pour mettre à jour les préférences dans OM2M.

CONCLUSION

Pour conclure, ce projet a été pour nous vraiment très intéressant, certainement le plus enrichissant depuis le début de notre cursus en école d'ingénieur. En effet, il nous a permis de développer notre autonomie, notre capacité d'auto-apprentissage, notre organisation et notre capacité à travailler en équipe. Grâce à ce projet, nous avons appris à utiliser de plusieurs logiciels, méthodes et langages de programmation. Nous avons également amélioré notre compréhension du fonctionnement d'applications faisant partie du domaine de l'Internet des Objets.

Concernant les perspectives de développement de ce projet, la reconnaissance faciale permettant d'identifier clairement une personne pourrait être un vrai plus et faisait partie de notre objectif initial. Malheureusement, nous n'avons pas réussi à effectuer la différence entre deux visages, mais nous avons quand même pu détecter différentes expressions du visage de manière efficace. Une autre perspective d'évolution serait de coder tous les contrôles de capteurs actionneurs en C++, notamment avec des classes et héritages, pour une meilleure modularité du code.

Le système final fonctionne donc bien, avec une interconnexion réussie : l'application contrôle les préférences de l'utilisateur qui utilise son visage (froncement des sourcils, bouche ouverte, sourire) pour contrôler les différents actionneurs de sa maison/appartement.

Cependant, nous regrettons d'avoir manqué un peu d'organisation et d'anticipation en demandant le matériel nécessaire dès le rendu de l'état de l'art, par exemple. Nous aurions pu aller plus loin et acquérir de nouvelles compétences.

Globalement, le projet est assez complet regroupant plusieurs spécificités : une application mobile, des capteurs, des actionneurs, un ESP, une carte Intel Edison, un logiciel de reconnaissance faciale avec des données partagées sur une même plateforme d'interconnexion : OM2M. Deux d'entre nous intégrant le PTP ISS en 5^{ème} année à l'INSA, connaître les bases de cette plateforme nous a semblé vraiment utile.

VERS UNE MAISON INTELLIGENTE...

RESUME :

Vivre de manière plus confortable, en contrôlant sa maison à l'aide de simples expressions du visage : allumer des lumières, faire tourner un ventilateur ou mettre en marche une cafetière avec un sourire. Un projet qui regroupe plateforme IoT, application Android, reconnaissance faciale et actionneurs/capteurs.

MOTS-CLES : *RECONNAISSANCE FACIALE – OBJETS CONNECTES – IoT – OM2M – APPLICATION ANDROID – CAPTEURS – HTTP – ESP – INTEL EDISON*

ABSTRACT :

To live a more comfortable life, by controlling your house with simple facial expressions: to be able to turn on a light, a fan, or a coffeemaker with a smile. A supervised project which includes IoT platform, Android application, facial recognition and actuators/sensors.

KEYWORDS : *FACIAL RECOGNITION – CONNECTED OBJECTS – IoT – OM2M – ANDROID APPLICATION – SENSORS – HTTP – ESP – INTEL EDISON*