# Rapport du projet

## Projet Pepper


Réalisé par :

| | | |
|---|---|---|
| M. ALYANAKIAN Théo | – | MT4 |
| M. BENCHIKH Hakim | – | IATIC4 |
| Mme. DOLIMAN Sarah | – | IATIC4 |
| Mme. DUREY Emilie | – | IATIC4 |
| M. HERBAYS Alexandre | – | MT4 |
| M. HOUDAYER Axel | – | IATIC4 |
| M. LALOUELLE Tanguy | – | SEE4 |
| M. MOITTIÉ Jordan | – | MT4 |
| M. RAGOT Anthonin | – | MT4 |
| M. WCISLO Tomasz | – | SEE4 |
| M. ZITOUNI Seiji | – | SEE4 |


Projet Inter-filières effectué du 24/10/2019 au 10/05/2020

# TABLE DES MATIERES

# TABLE DES ILLUSTRATIONS

# I. Introduction

Dans le cadre d'un projet inter-filière proposé par l'ISTY, les spécialités informatique, systèmes embarqués et mécatronique ont été amenées à travailler ensemble. L'objectif de ce projet est la prise en main du robot Pepper, développé par la société SoftBank Robotics, en utilisant la plateforme de développement logicielle ROS (Robot Operating System) ainsi que les différents SDK mis à disposition par la société.


# II. Définition du besoin

Avant de réfléchir à l'aspect technique, nous devons d'abord réfléchir aux actions que doit réaliser Pepper : scénarios que l'on veut mettre en place par le robot.

Le besoin initial est ici d'effectuer une visite guidée de l'ISTY avec Pepper en permettant son pilotage en mode manuel avec une manette ou une interface homme-machine (IHM) ou bien avec un mode plus autonome en suivant un parcours au sol. Le robot devra éventuellement interagir avec des humains en les identifiant par reconnaissance faciale et en comprenant des gestes.

Dans un second temps, nous allons établir les étapes suivantes du scénario afin de décrire les successions de déplacements/actions du robot dans le but de faire la visite :

- Initiation de la visite avec un visiteur : Pepper dira "bonjour, bienvenue à l'ISTY, je vais vous faire visiter l'établissement [...]" avec une animation affichée sur la tablette.
- Il y aura une visite par étage ou par "pièce", pour éviter que Pepper ne soit contraint d'appuyer sur des boutons pour l'ascenseur, ouvrir des portes, etc…
- Séquence de déplacements avec arrêts pour expliquer le contexte et l'endroit sur lequel on est, exemple :
    - Pepper à l'accueil, demandera aux visiteurs de le suivre jusqu'à un endroit, s'arrêtera, et dira : "ici se trouve la machine à café, c'est ici que vous pouvez vous abreuver entre deux cours"
    - Ensuite, Pepper ira à l'autre bout de la pièce (ou de l'étage) pour présenter les autres parties des lieux.

# III. Avancement du projet au 26/11/19

À la date du 26 novembre, nous avons réussi à :

- Obtenir la majorité des ordinateurs opérationnels pour utiliser ROS Indigo et les différents SDK.
- Utiliser *RQT_IMAGE_VIEW* pour obtenir les différentes sorties vidéo du robot sur ses nombreuses caméras, de différents types.
- Afficher des animations sur la tablette de Pepper telles qu'un site web ou une image via l'utilisation des WebSockets en langage JavaScript.
- Utiliser la fonction Say pour faire parler Pepper via des fichiers C++ et en JavaScript via les WebSockets.
- Faire mouvoir le robot en ligne de commande grâce à ROS Indigo.
- Créer une télécommande en JavaScript pour contrôler le robot Pepper.
- Afficher le nombre de visages détectés par le robot Pepper.

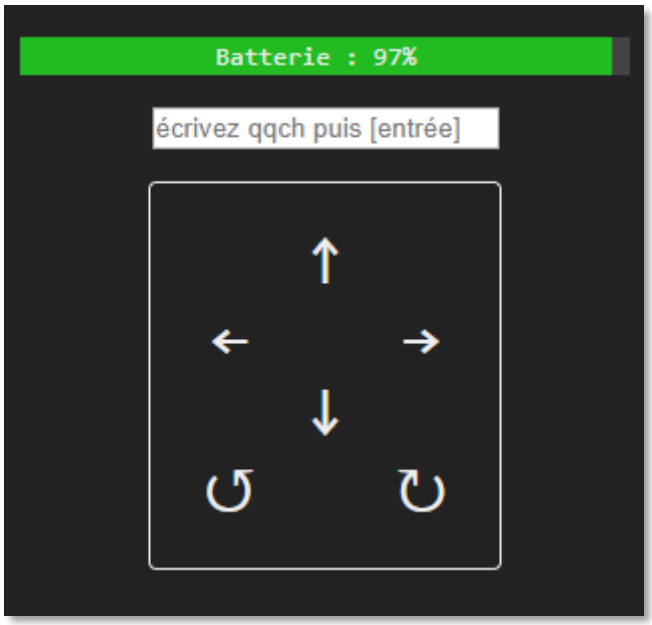



**Figure 1: Télécommande JavaScript pour commander Pepper**

# IV. Inventaire des missions

Dans l'optique de la réalisation du projet, nous avons défini plusieurs objectifs techniques à atteindre tout au long du projet, à savoir :

| ***Missions à effectuer*** | ***Priorité*** |
|---|---|
| Créer une application .apk sous Unity permettant de faire fonctionner Pepper en utilisant des blocs | Moyenne |
| Capitaliser l'ensemble des documents sur Pepper, et créer des procédures (Utilisation de Pepper, Installation de ROS, Installation de .APK) | Moyenne |
| Contrôler Pepper avec une manette à Joystick | Moyenne |
| Utiliser le retour d'image et la reconnaissance faciale de Pepper afin de reconnaître les personnes en face de Pepper (en C++ et/ou JavaScript) | Haute |
| Utiliser les capteurs de positionnement de Pepper afin d'éviter des obstacles lors de ses déplacements (en C++ et/ou JavaScript) | Haute |
| Actionner les bras et la tête de Pepper lorsqu'il communique ou pour serrer la main d'un individu (en C++) | Haute |
| Affecter des réponses types à Pepper lorsqu'on communique avec lui (en C++ et/ou JS) afin de pouvoir tenir une communication avec lui | Haute |
| Effectuer une vidéo de présentation de l'ensemble des fonctionnalités que nous avons créées pour Pepper | Haute |
| Implémenter une architecture ROS et utiliser OpenCV sur Pepper | Haute |
| Simuler l'ensemble des fonctions sous Gazebo et effectuer une comparaison entre la simulation et la réalité | Moyenne |
| Afficher des animations et/ou une web app sur la tablette de Pepper (en JS) | Haute |

# V. Contrôle des déplacements du robot

## A. Télécommande HTML/JavaScript

Pour contrôler le robot, nous avons décidé d'utiliser les technologies du web (HTML5/CSS3/JavaScript) pour leur portabilité et leur simplicité de mise en place. En effet, tous nos appareils sont aujourd'hui équipés d'un navigateur internet et donc sont capables d'interpréter du JavaScript, qui d'ailleurs ne nécessite aucune compilation. Cela signifie que la télécommande sera accessible à partir d'un ordinateur, d'un smartphone ou encore d'une tablette sans avoir à installer quoi que ce soit.

**Figure 2 : Logos des navigateurs Safari, Firefox, Chrome, Edge et Opera compatibles avec notre télécommande**

Nous avons tout d'abord créé un fichier « remote-control.html » avec une interface basique composée de 4 flèches pour contrôler les déplacements du robot, 2 flèches pour faire tourner le robot, une barre indiquant la charge de la batterie en pourcentage et un champ de texte pour faire parler le robot.

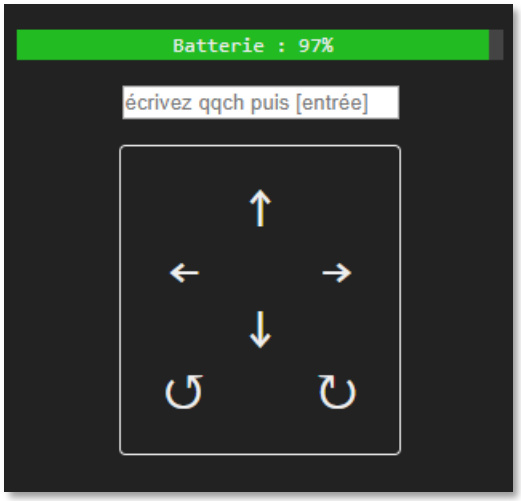



**Figure 3 : Télécommande Pepper**

Dans un second temps, il faut inclure le SDK JavaScript NAOqi, compatible avec les robots Nao et Pepper. Pour ce faire, on rajoute la ligne suivante avant la balise fermante </head> :

```
<script src="http://192.168.43.2/libs/qi/2/qi.js"></script>
```

*Remarque : le script se trouve dans le système de fichiers de Pepper, d'où l'adresse IP du robot dans l'URL*

Une fois le SDK inclus, on peut accéder à la fonction QiSession qui permet de se connecter à un robot Pepper via les WebSockets, assurant ainsi une communication bidirectionnelle avec le robot. En réalité, NAOqi utilise la bibliothèque Socket.IO qui constitue une couche d'abstraction supplémentaire et permet de découper les trames en messages.

La fonction QiSession prend 3 paramètres :

- Un callback (fonction de rappel) qui sera appelé lorsque la connexion a été établie.
- Un callback pour la déconnexion.
- L'adresse IP du robot.

Une fois que la connexion à Pepper a été établie, le callback de connexion est appelé avec, en premier paramètre, un objet représentant la session en cours. Il est nécessaire de conserver une référence à cet objet car il sera utilisé par la suite pour accéder aux différentes APIs/services de Pepper (API de vision, API de gestion du mouvement, etc).

Nous avons donc créé la fonction PepperRobot pour gérer la connexion à Pepper et l'initialisation des services :

```
function PepperRobot(robotIp, services, onReady) {
    QiSession(function (session) {
        console.log("QiSession: connected");
        this.session = session;
        this.initServices(services);
    }.bind(this), function () {
        console.log("QiSession: disconnected");
    }, robotIp);

    this.initServices = function(list) {
        if(list.length === 0)
            return;

        var serviceIndex = 0, initService = function(name) {
            if(!name) {
                onReady();
                return;
            }
            console.log('init service: ' + name);
            this.session.service(name).then(function(service) {
                this[name] = service;
                initService(list[++serviceIndex]);
            }.bind(this));
        }.bind(this);

        initService(list[serviceIndex]);
    };
};
```

Nous pouvons ensuite instancier un objet de type PepperRobot, à l'aide du mot clé *new*. En JavaScript, une fonction peut être instanciée et être utilisée comme un objet. A partir de la version ECMAScript 2015 du JavaScript, il est également possible d'utiliser le mot clé *class* pour faire de la programmation orientée objet. Néanmoins, les anciens navigateurs comme Internet Explorer 11 ne supportent pas ce mot clé. Pour une compatibilité avec un maximum de navigateurs, nous n'utiliserons pas les nouvelles fonctionnalités du JavaScript.

Pour créer une instance de PepperRobot, 3 paramètres sont nécessaires :

- L'IP du robot.
- La liste des services à initialiser pour une utilisation ultérieure.
- Un callback qui sera appelé une fois que tous les services ont été initialisés.

Créons une fonction appelée *main* qui sera appelée une fois que les services sont prêts et puis instancions un objet de la classe PepperRobot :

```
function main() {
    console.log('Pepper is ready!');
}

var pepper = new PepperRobot('192.168.43.2', [
    'ALMotion',
    'ALMemory',
    'ALBattery',
    'ALRobotPosture',
    'ALTextToSpeech',
    'ALTabletService',
    'ALFaceDetection',
    'ALBehaviorManager',
    'ALPeoplePerception',
    'ALVisionRecognition',
    'ALVideoDevice',
    'ALLandMarkDetection',
    'ALAudioDevice',
    'ALSystem',
    'ALCloseObjectDetection',
    'ALAutonomousLife'
], main);
```

Les noms des services sont assez explicites mais, pour donner un exemple, ALMotion est celui qui permet de déplacer le robot et gérer le mouvement de ses membres et de la tête.

Une fois que tout a été chargé correctement, la fonction *main* sera appelée et on pourra voir le message « Pepper is ready ! » dans la console JavaScript du navigateur. Pour accéder aux outils de développement du navigateur, le raccourci le plus courant est F12. Certains requièrent néanmoins un Ctrl+Maj+I pour ouvrir la console. Il faut ensuite cliquer sur l'onglet « Console » de la fenêtre qui s'est ouverte.

A partir de maintenant, tous les services chargés sont accessibles en tapant *pepper.NomDuService* dans la console !

Pour faire dire « Bonjour » à Pepper, il faut utiliser la méthode *say* du service *ALTextToSpeech*. Il est donc possible d'écrire *pepper.ALTextToSpeech.say('Bonjour')* dans la console ou d'utiliser le champ de la télécommande prévu à cet effet.

Nous allons maintenant découvrir comment faire avancer Pepper, le faire reculer ou encore le faire tourner sur lui-même !

Pour déplacer Pepper ou faire bouger un de ses membres, nous allons nous servir du service ALMotion.

Il existe deux fonctions pour faire se déplacer Pepper : *move* et *moveTo*. Avec la fonction *moveTo*, on peut déplacer le robot d'une distance donnée sur les axes X et Y et le faire tourner d'un certain angle.

Avec *move*, on le fait se déplacer à une vitesse donnée puis on l'arrête avec *stopMove*. Sans l'appel à *stopMove* le robot avancerait jusqu'à rencontrer un obstacle, si bien sûr la détection d'obstacle est activée. Prudence !

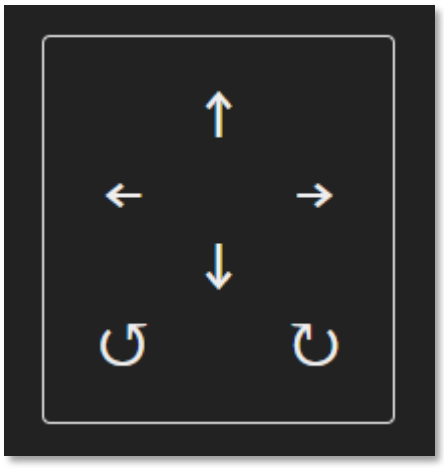


**Figure 4 : Boutons permettant de déplacer Pepper**

Voici le code pour faire avancer et reculer Pepper :

```
var moveArrows = document.querySelectorAll('.move-pepper-arrows .arrow');
var stopMove = function() {
    pepper.ALMotion.stopMove();
}
// forward
moveArrows[0].addEventListener('mousedown', function() {
    pepper.ALMotion.move(1, 0, 0); // move(x, y, theta)
}, false);
moveArrows[0].addEventListener('mouseup', stopMove, false);
// backward
moveArrows[3].addEventListener('mousedown', function() {
    pepper.ALMotion.move(-1, 0, 0);
}, false);
moveArrows[3].addEventListener('mouseup', stopMove, false);
```

*pepper.ALMotion.move(1, 0, 0);* permet de faire avancer Pepper à une vitesse de 1 m/s.

## B. Manette Xbox/PlayStation

Pour contrôler Pepper de façon plus naturelle, nous avons décidé d'utiliser une manette de PlayStation connectée à un ordinateur. Pour y accéder à partir du navigateur, nous avons dû utiliser la « Gamepad API » en JavaScript, supportée par tous les navigateurs récents.

Pour utiliser une manette de jeux en JavaScript, il faut tout d'abord identifier les manettes connectées. Il y a deux façons de procéder :

- Si la propriété *ongamepadconnected* existe dans l'objet *window* du navigateur, cela signifie que l'on peut souscrire à cet évènement et être informé de la connexion d'une nouvelle manette à l'ordinateur
- Si l'évènement décrit ci-dessus n'est pas disponible, il faut scanner les gamepads disponibles manuellement.

Voici le code permettant de « trouver » tous les gamepads connectés :

```
var hasEvents = 'ongamepadconnected' in window;

function connecthandler(e) {
    addgamepad(e.gamepad);
}

function scangamepads() {
    var gamepads = navigator.getGamepads ? navigator.getGamepads() : (navigator.webkitGetG
amepads ? navigator.webkitGetGamepads() : []);
    for (var i = 0; i < gamepads.length; i++) {
        if (gamepads[i]) {
            if (!(gamepads[i].index in controllers))
                addgamepad(gamepads[i]);
            else
                controllers[gamepads[i].index] = gamepads[i];
        }
    }
}

if (!hasEvents) {
    setInterval(scangamepads, 500);
} else {
    window.addEventListener("gamepadconnected", connecthandler);
}
```

Lorsque le code ci-dessus a fini par trouver un gamepad, la fonction suivante est appelée :

```
function addgamepad(gamepad) {
    controllers[gamepad.index] = gamepad;
    // ...
    requestAnimationFrame(updateStatus);
}
```

Elle va à son tour appeler la function *updateStatus* qui va se rappeler en boucle pour obtenir l'état du gamepad.

La fonction *updateStatus* scanne l'état des gamepads connectés et s'appele à nouveau.

```
var keys = [], prevKeys = [];

function updateStatus() {
    scangamepads();
    keys = [];
    // 1. on scanne les touches de la manette
    // 2. si l'état d'une touche a changé, on appelle la fonction handleEvent()
    // ...
    prevKeys = keys;
    requestAnimationFrame(updateStatus);
}
```

Et voici un bout de la fonction *handleEvent*, qui vérifie si une touche a été pressée ou bien relâchée :

```
function handleEvent(i, curr, prev) {
    if(i == 7) { // On avance --> Gamepad R2
        if(curr.includes(i) && !prev.includes(i)) {
            console.log("R2 down")
            pepper.ALMotion.move(1, 0, 0);
        } else if(!curr.includes(i) && prev.includes(i)) {
            console.log("R2 up")
            pepper.ALMotion.stopMove();
        }
    }
    // ...
}
```

Comme pour la télécommande présentée précédemment, on utilise les fonctions *move* et *stopMove* pour faire avancer et arrêter Pepper (respectivement).


**Figure 5 : Manette PS4 avec commandes pour Pepper**

## C. ROS

Nous avons commencé par utiliser le langage ROS avant d'utiliser d'autres langages pour contrôler le robot Pepper. En effet, nous avons installé la version ROS Indigo puis à l'aide du terminal nous avons lancé des commandes pour faire avancer le robot.

ROS ayant besoin du SDK Python (ou C++) NAOqi, nous avons dû le télécharger puis l'ajouter dans le PATH.

Ensuite, il suffit de lancer un terminal avec la commande suivante :

roslaunch pepper_bringup pepper_full_py.launch nao_ip : = 192.168.1.4 roscore_ip : = <roscore_ip>

Où roscore_ip est l'adresse IP de l'ordinateur. Cette commande permet de lancer un serveur ROS qui communiquera avec le robot grâce à son adresse IP, il faut donc que le robot Pepper et l'ordinateur qui commande soient connectés sur le même réseau.

Il faut laisser ce terminal tourner en fond, et ouvrir un autre terminal pour publier un « Node ».

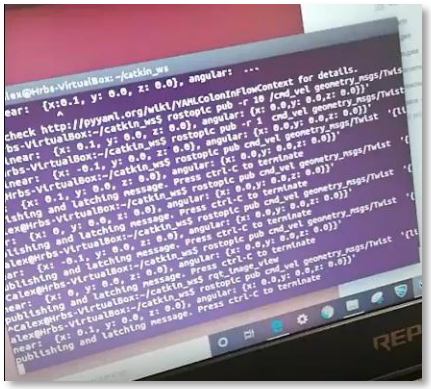


**Figure 6 : Terminal avec ROS**

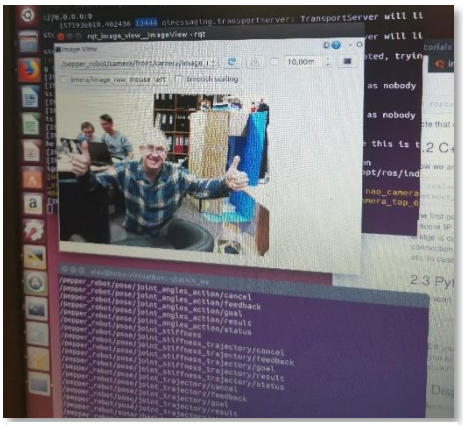


**Figure 7 : Utilisation des caméras avec ROS grâce à rqt_image_view**

Pour faire déplacer Pepper, il faut taper les requêtes suivantes :

- **Reculer** :
  rostopic pub cmd_vel geometry_msgs / Twist '{near: {x: -0.1, y: 0.0, z: 0.0}, angular: {x: 0.0,y: 0.0,z: 0.0}}'

- **Stop** :
  rostopic pub cmd_vel geometry_msgs / Twist '{near: {x: 0, y: 0.0, z: 0.0}, angular: {x: 0.0,y: 0.0,z: 0.0}}'

- **Avancer** :
  rostopic pub cmd_vel geometry_msgs / Twist '{linear: {x: 0.1, y: 0.0, z: 0.0}, angular: {x: 0.0,y: 0.0,z: 0.0}}'

Par manque de temps nous n'avons pas pu investiguer plus dans ce langage et dans la publication des nœuds en C++ ou Python.

# VI. Utilisation des fonctionnalités multimédia de Pepper

## A. Tablette numérique

Pepper nous offre des possibilités multimédia grâce à sa tablette numérique fixée sur son torse ainsi que ses différents haut-parleurs.

Nous avons donc imaginé au début du projet un scénario utilisant la tablette numérique, ce scénario était le suivant :

*« Lors d'une Journée Portes Ouvertes, le robot Pepper accueille les futurs étudiant(e)s avec une reconnaissance vocale et le site de l'ISTY ou une application dédiée à la visite de l'établissement sur la tablette numérique. Les futurs étudiant(e)s pourront alors interagir avec le robot par différents moyens : vocal ou par la tablette. […] »*

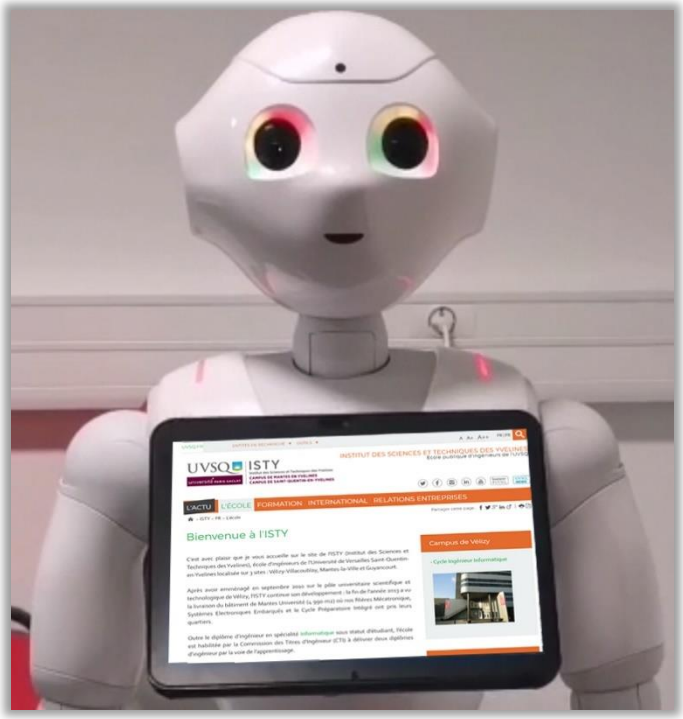



**Figure 8 : Pepper avec le site de l'ISTY**

Grâce au SDK NAOqi fourni par le fabricant, nous avons pu interagir avec la tablette.

Nous avons décidé d'utiliser le JavaScript pour lancer un site web sur la tablette (nous aurions pu utiliser Python ou C++ puisque c'est le même SDK), pour cela nous avons manipulé ses fonctions avec l'outil console (F12 sur Chrome) puis nous avons inséré le code dans le fichier « remote-control.html ».

En effet en regardant la documentation du SDK (voir Annexe/Bibliographie) nous avons constaté qu'il y avait une API 'ALTabletService' qui permet de prendre le contrôle de la tablette, ce « service » permet d'effectuer plusieurs actions sur la tablette comme : Afficher un site web, mettre en place un vidéo player, afficher une image, afficher une boite de dialogue, afficher les paramètres du robot ….

Nous nous sommes intéressés plus particulièrement à la méthode « showWebview » qui permet d'afficher un site web sur la tablette, nous avons donc créé un bouton sur le « remote-control.html » pour afficher le site web de l'ISTY.


**Figure 9 : Télécommande JavaScript pour contrôler Pepper avec boutons « afficher le site de l'ISTY »**

Voici la méthode utilisée en JavaScript :

```
document.getElementById('show-webview').addEventListener('click', function() {
    pepper.ALTabletService.showWebview('http://www.isty.uvsq.fr/');
}, false);
```

Comme dit précédemment, nous utilisons « pepper » qui est une instance de « PepperRobot » puis nous utilisons la méthode showWebview du service ALTabletService et nous rentrons comme argument l'adresse du site web de l'ISTY.

La prochaine étape aurait été de créer une application dédiée (soit en web App, soit une application Android) à la visite de l'école et on aurait affiché cette application sur la tablette de Pepper.

Cette application aurait donné des informations sur l'école, sur la pièce / l'étage où Pepper est en train de faire visiter les futurs étudiant(e)s.

Elle aurait aussi permis l'enregistrement des visages pour que Pepper puisse reconnaître la personne qui est en face de lui.

## B. Haut-parleurs

Nous avons vu précédemment qu'il était possible de faire parler Pepper avec l'API ALTextToSpeech. Et bien ce n'est pas tout ! Il est également possible de lire un fichier audio et produire un signal audio sinusoïdal à une certaine fréquence.

Pour lire un fichier audio, nous utilisons l'API ALAudioPlayer. Voici quelques fonctions dont on peut se servir :

- *loadFile("chemin/du/fichier")* : charge un fichier audio (un .wav par exemple)
- *playFile* : charge et lit un fichier audio
- *play* : lit un fichier audio chargé préalablement.

Découvrons maintenant l'API ALAudioDevice, qui permet de régler le volume des haut-parleurs de Pepper, d'enregistrer du son grâce aux microphones intégrés ou encore de jouer des notes de musique.

Nous allons nous intéresser à la génération d'un signal audio sinusoïdal pour jouer des notes de musique. Pour ce faire, voyons à quoi ressemble le prototype de la fonction *playSine* :

*playSine(frequency, volume, duration)*

Nous avons donc besoin de 3 paramètres : la fréquence de la note, son volume et sa durée. Pour pouvoir écrire les notes en notation anglo-saxonne dans notre programme, nous devons convertir le nom de chaque note en fréquence. Les lettres A B C D E F G en notation anglo-saxonne sont l'équivalent des notes que nous connaissons : La Si Do Ré Mi Fa Sol.

Voici les notes que nous aimerions pouvoir jouer et leurs fréquences en Hertz (Hz) :

Note Frequency Chart

| | Octave 0 | Octave 1 | Octave 2 | Octave 3 | Octave 4 | Octave 5 | Octave 6 | Octave 7 | Octave 8 |
|---|---|---|---|---|---|---|---|---|---|
| C | 16.35 | 32.70 | 65.41 | 130.81 | 261.63 | 523.25 | 1046.50 | 2093.00 | 4186.01 |
| C♯ | 17.32 | 34.65 | 69.30 | 138.59 | 277.18 | 554.37 | 1108.73 | 2217.46 | 4434.92 |
| D | 18.35 | 36.71 | 73.42 | 146.83 | 293.66 | 587.33 | 1174.66 | 2349.32 | 4698.64 |
| D♯ | 19.45 | 38.89 | 77.78 | 155.56 | 311.13 | 622.25 | 1244.51 | 2489.02 | 4978.03 |
| E | 20.60 | 41.20 | 82.41 | 164.81 | 329.63 | 659.26 | 1318.51 | 2637.02 | 5274.04 |
| F | 21.83 | 43.65 | 87.31 | 174.61 | 349.23 | 698.46 | 1396.91 | 2793.83 | 5587.65 |
| F♯ | 23.12 | 46.25 | 92.50 | 185.00 | 369.99 | 739.99 | 1479.98 | 2959.96 | 5919.91 |
| G | 24.50 | 49.00 | 98.00 | 196.00 | 392.00 | 783.99 | 1567.98 | 3135.96 | 6271.93 |
| G♯ | 25.96 | 51.91 | 103.83 | 207.65 | 415.30 | 830.61 | 1661.22 | 3322.44 | 6644.88 |
| A | 27.50 | 55.00 | 110.00 | 220.00 | 440.00 | 880.00 | 1760.00 | 3520.00 | 7040.00 |
| A♯ | 29.14 | 58.27 | 116.54 | 233.08 | 466.16 | 932.33 | 1864.66 | 3729.31 | 7458.62 |
| B | 30.87 | 61.74 | 123.47 | 246.94 | 493.88 | 987.77 | 1975.53 | 3951.07 | 7902.13 |

**Figure 10 : Les notes de musique et leurs fréquences**

On observe que la fréquence du Do (C) à l'octave 1 est égale à 2 fois la fréquence à l'octave 0.

A l'octave 2, elle est égale à 4 fois la fréquence à l'octave 0 et à l'octave 3, 8 fois la fréquence à l'octave 0, etc.

On peut donc écrire la formule suivante pour obtenir la fréquence d'une note à une octave donnée :

$$f_{octave} = f_{octave0} \times 2^{octave}$$

Créons une fonction permettant d'effectuer la conversion :

```
function noteToFreq(str) {
    var notes = {
        'c': 16.35,
        'c#': 17.32,
        'cs': 17.32,
        'df': 17.32,
        'd': 18.35,
        'd#': 19.45,
        'ds': 19.45,
        'ef': 19.45,
        'e': 20.60,
        'ff': 20.60,
        'e#': 21.83,
        'f': 21.83,
        'f#': 23.12,
        'fs': 23.12,
        'gf': 23.12,
        'g': 24.50,
        'g#': 25.96,
        'gs': 25.96,
        'af': 25.96,
        'a': 27.50,
        'a#': 29.14,
        'as': 29.14,
        'b': 30.87,
        'cf': 30.87
    };

    var note = str.replace(/[^a-gs#]/gi, '').slice(0, 2).toLowerCase();
    var octave = parseInt(str.replace(/[^0-8]/g, ''));
    octave = isNaN(octave) ? 0 : octave;

    return notes[note] * Math.pow(2, octave);
}
```

*Exemple : Do dièse = C# = C sharp. On peut donc écrire C# ou Cs dans notre programme. Do bémol = C flat donc Cf*

Voici le code permettant de jouer une suite de notes :

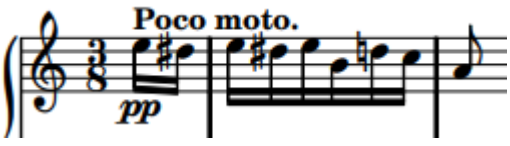
```
function playNotes(volume, notes) {
    var i = 0;

    var playNextNote = function() {
        if(i >= notes.length)
            return;

        if(notes[i].length === 2)
            pepper.ALAudioDevice.playSine(noteToFreq(notes[i][0]), volume, 0, notes[i][1])
;

        setTimeout(playNextNote, notes[i++][1] * 1000 + 50);
    };

    playNextNote();
}
```

Le premier paramètre est le volume, de 0 à 100 et le second est un tableau de tableaux sous la forme suivante : [note,fréq]. *Remarque : le +50 dans l'appel à setTimeout permet de compenser les lags du réseau, car il ne faut pas oublier qu'à chaque appel à une API de Pepper, une trame est envoyée au robot via les WebSockets, qui va ensuite exécuter l'action demandée.*

Exemple : pour jouer le début de Lettre à Élise, de Beethoven :

```
playNotes(50, [['e5',0.2],['d#5',0.2],['e5',0.2],['d#5',0.2],['e5',0.2],['b4',0.2],['d5',0
.2],['c5',0.2],['a4',0.5]]);
```

Ce qui correspond à cet extrait de la partition :



# VII. Reconnaissance faciale

## A. En utilisant les APIs

A la dernière séance du projet inter-filière, nous avons pu mettre en place une reconnaissance des visages par Pepper grâce à l'utilisation de certains « services » et « méthodes » qui sont inclus dans ses services.

En effet, le Software Kit Développement fourni par Soft Bank propose le service « ALMemory » qui permet de souscrire à des évènements du robot sans passer par les services spécifiques à chaque fonctionnalité (AL...), il s'utilise comme ceci : `pepper.ALMemory.subscriber('FaceDetected').then(function(subscriber) { … })`

Ici, au lieu d'utiliser le service « ALFaceDetection » on utilise ALMemory qui souscrit au service « FaceDetected ». En effet, ce service permet, à partir des visages enregistrés par Pepper, de reconnaître une personne.

Nous allons voir en détails l'enregistrement des visages par Pepper à travers le service « ALFaceDetection » puis la reconnaissance de ces visages par Pepper.

### 1. Enregistrement des visages

Pour enregistrer un visage nous avons mis en place un bouton sur le fichier « remote-control.html » qui permet d'enregistrer un visage par Pepper en utilisant le service « ALFaceDetection » et qui demande le prénom de la personne, ensuite Pepper stockera ces informations dans sa mémoire.

```
//learn face
document.getElementById('learn-face').addEventListener('click', function() {
    var name = prompt('Entrez le nom de la personne :');
    if(name.length < 2) {
        alert('Vous devez entrer au moins 2 caractères !');
        return;
    }
    if(confirm('Veuillez vous placer devant le robot, puis appuyez sur OK'))
        pepper.ALFaceDetection.learnFace(name);
    else
        alert('Vous avez annulé l\'opération');
}, false);
```

**Figure 11 : Code pour apprendre un visage avec l'API (lignes 228 à 238 de remote-control.html)**

Comme vous pouvez le remarquer nous demandons à l'utilisateur d'entrer le nom de la personne qui veut enregistrer son visage, puis nous utilisons la méthode « learnFace » du service « ALFaceDetection » avec en argument le nom de la personne pour apprendre à Pepper à associer le visage avec le nom de la personne.

Une fois que le processus est lancé, la personne à 5 secondes pour placer son visage devant le robot, puis durant le processus d'apprentissage les yeux du robot deviennent bleus.

Les yeux du robot deviennent verts en moins d'une seconde si le visage est vu par Pepper dans les conditions optimales (pas d'ombre partielle sur le visage …) si les yeux sont toujours bleus après quelques secondes, la personne doit se repositionner afin de changer les conditions d'apprentissage.

## 2. Reconnaissance des visages

Maintenant que les visages sont enregistrés dans la mémoire de Pepper, on peut utiliser la reconnaissance faciale de Pepper, par exemple pour saluer un(e) étudiant(e) / professeur avec son prénom.

```
//facial recognition
var lastPerson = '';

pepper.ALMemory.subscriber('FaceDetected').then(function(subscriber) {
    subscriber.signal.connect(function(info) {
        // console.log(info);  //uncomment to display info on detected faces
        try {
            var personName = info[1][0][1][2];

            if(personName) {
                console.log('face detected: ' + personName);

                if(personName != lastPerson) {
                    pepper.ALTextToSpeech.say('Bonjour ' + personName);
                    lastPerson = personName;
                }
            }
        } catch(e) {}

        console.log('Nombre de visages détectés : ' + ((info.length > 0) ? info[1].length - 1 : 0));
    });
});
pepper.ALFaceDetection.subscribe('FaceDetected');
```

**Figure 12 : Code pour reconnaître les visages (lignes 200 à 221 de remote-control.html)**

**ALFaceDetection** est donc un module de vision dans lequel le robot essaie de détecter et de reconnaître les visages devant lui, ce module est basé sur une solution de détection/reconnaissance de visage fournie par OMRON.

Lorsque le robot détecte un visage, il a sa position ainsi qu'une liste de coordonnées angulaires pour les caractéristiques des visages (yeux, nez et bouche).

La fonction de reconnaissance renvoie pour chaque image le nom des personnes reconnues qui est stocké à une certaine position dans le tableau « info » : `var personName = info[1][0][1][2];`

En effet, l'événement FaceDetected() renvoie une valeur organisée comme ceci :

```
FaceDetected  =
[
  TimeStamp ,
  [  FaceInfo [ N ],  Time_Filtered_Reco_Info  ],
  CameraPose_InTorsoFrame ,
  CameraPose_InRobotFrame ,
  Camera_Id
]
```

Il y a donc 5 champs dans le tableau, qui sont eux même des tableaux.

**TimeStamp** est le champ contenant l'horodatage de l'image qui a été utilisée pour effectuer la détection. Il est composé comme ceci :

```
TimeStamp  =
[
  TimeStamp_Seconds ,
  Timestamp_Microseconds
]
```

Ici le champ qui nous intéresse le plus est **FaceInfo**. Il est composé de deux champs, un champ « **ShapeInfo** » et un champ « **ExtraInfo[N]** »**.**

Le champ **ShapeInfo** contient les informations sur le visage détecté :

```
ShapeInfo  =
[
  0 ,
  alpha ,
  beta ,
  sizeX ,
  sizeY
]
```

- **Alpha** et **bêta** représentent l'emplacement du visage en termes d'angles de caméra.
- **sizeX** et **sizeY** sont la taille du visage dans l'angle de la caméra.

Dans **ExtraInfo** on a :

- **faceID** : qui représente le numéro d'identification du visage.
- **scoreReco** : qui est le score renvoyé par le processus de reconnaissance (plus il est haut mieux c'est).
- **faceLabel** : qui renvoie le nom du visage reconnu si le visage a été reconnu.
- **leftEyePoints** et **rightEyePoints** fournissent les positions des points pour les yeux.
- **nosePoints** et **mouthPoints** fournissent respectivement les positions des points pour le nez et les lèvres.

```
ExtraInfo =
[
  faceID,
  scoreReco,
  faceLabel,
  leftEyePoints,
  rightEyePoints,
  unused, # for backward-compatibility issues
  unused,
  nosePoints,
  mouthPoints
]
```

Ici, dans notre programme pour saluer une personne reconnue, nous devons récupérer la valeur contenue dans **faceLabel**. Pour cela, nous devons dans le tableau retourné par FaceDetected, choisir le champ **FaceInfo[N]** du sous tableau qui est à la **positon 1**. Ce champ est donc à la **position 0** du sous tableau. Puis nous choisissons le champ **ExtraInfo[N]** qui est à la **position 1.** Enfin dans ce tableau, nous prenons la valeur contenue à la **position 2** qui est la valeur du champ **faceLabel**. On obtient alors : `var personName = info[1][0][1][2];`

Une fois cette valeur récupérée, nous faisons appel à la méthode « say » du service `'ALTextToSpeech'` pour que Pepper salue la personne qu'il a reconnue par exemple : « Bonjour Mr Bonnin » ou « Bonjour Mr Blazevic ».

Ensuite, nous paramétrons la variable « lastPerson » avec le nom du visage qu'il vient de détecter pour éviter que Pepper ne répète constamment la phrase grâce à la condition : `if(personName != lastPerson)` .

Nous avons aussi rajouté un bouton sur « remote-control.html » pour lister les visages que Pepper connait grâce à la méthode « getLearnedFacesList() » du service `'ALFaceDetection'`.

## B. En utilisant OpenCV sous Windows

N'ayant pas eu le temps de tester OpenCV avec Pepper à cause d'un confinement imprévu, nous avons décidé d'effectuer de la reconnaissance faciale sous Windows.

Avant de commencer à développer, il faut compiler et installer OpenCV. Pour ce faire, nous avons téléchargé à partir de GitHub le code source d'OpenCV et les modules additionnels « opencv_contrib », parmi lesquels figure le module « face » permettant de faire de la reconnaissance faciale. Après avoir extrait le code source d'OpenCV dans un dossier vide, nous l'avons ouvert avec Visual Studio puis nous avons configuré le projet avec CMake. Nous avons désactivé le module CUDA et les modules s'y rapportant pour faciliter et accélérer la compilation de la bibliothèque.

Nous avons également choisi de compiler OpenCV avec le flag /MT, qui signifie que le CRT (C Run-Time) de Microsoft sera linké statiquement à la bibliothèque. Pour rappel, les flags suivants sont disponibles sous Windows avec le compilateur MSVC intégré à Visual Studio :

- */MT (Multi-threaded) :* le CRT statique est utilisé lors du linkage
- */MD (Multi-threaded DLL) :* l'application est linkée au CRT dynamique et nécessitera une DLL système pour fonctionner
- */MTd (Multi-threaded Debug) :* équivalent de /MT en mode Debug
- */MDd (Multi-threaded Debug DLL) :* équivalent de /MD en mode Debug.

*Attention ! Si vous compilez votre application avec /MT et que vous linkez une bibliothèque qui a été compilée avec /MD (et inversement), l'édition de liens échouera ! L'application doit impérativement être linkée au même CRT que les bibliothèques qu'elle utilise.*

Par ailleurs, nous avons compilé OpenCV pour l'architecture **x64**, ce qui signifie que notre application devra aussi être compilée en 64 bits et ne fonctionnera pas sur un ordinateur 32 bits.

Nous pouvons maintenant créer un nouveau projet C++ avec Visual Studio et utiliser la bibliothèque OpenCV pour faire de la détection de visages et de la reconnaissance faciale.

La détection de visages consiste à « trouver » les visages sur une image.

La reconnaissance faciale, elle, a pour objectif de différencier les visages de plusieurs personnes. On peut s'en servir par exemple pour créer un système d'authentification par reconnaissance faciale. Ainsi, Jean-Philippe pourra déverrouiller son téléphone sans même devoir saisir de mot de passe !

Nous allons donc combiner ces deux notions pour reconnaître les visages en temps réel sur un flux vidéo !

## 1. Détection de visages avec les cascades OpenCV

Pour comprendre ce qu'est une cascade de classificateurs, il faut tout d'abord définir le terme de classificateur.

Il s'agit d'une méthode consistant à parcourir l'image à l'aide d'une fenêtre glissante de taille fixe et de déterminer si un visage s'y trouve. Il existe deux principaux classificateurs disponibles dans OpenCV : Haar et LBP.

### a) Classificateur Haar

Pour savoir si un visage se trouve dans une certaine zone, il faut identifier un certain nombre de caractéristiques ou features en anglais (bords, coins, lignes, centres/pourtours) puis appliquer un certain nombre de calculs.

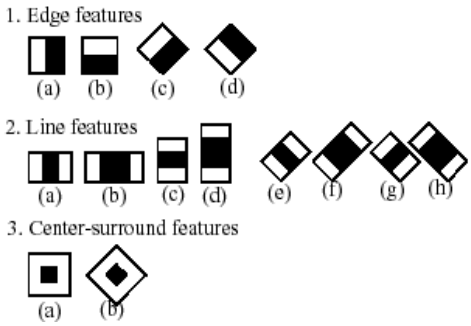



**Figure 13 : Caractéristiques recherchées par le classificateur Haar**

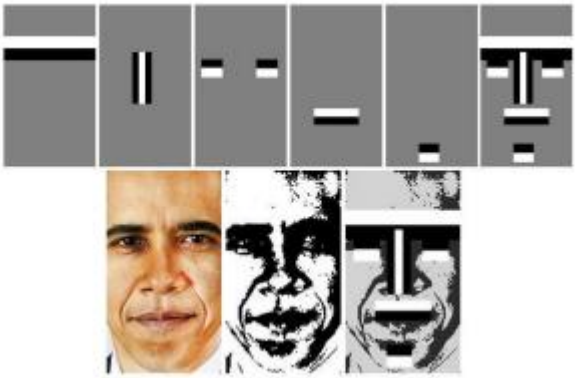


**Figure 14 : Caractéristiques Haar du visage de Barack Obama**

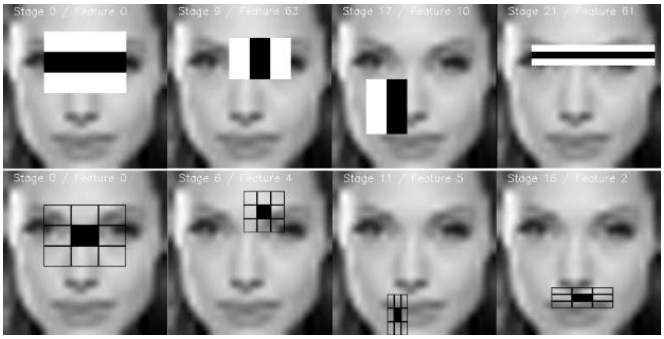


**Figure 15 : Caractéristiques Haar du visage d'Angelina Jolie**

Bien sûr, les photos ci-dessus ne sont qu'un aperçu des caractéristiques des visages d'Angelina Jolie et de Barack Obama. L'algorithme est capable d'en trouver bien plus, allant de quelques centaines à quelques milliers…

### b) Classificateur LBP (Local Binary Patterns)

Cet algorithme est différent du premier puisqu'il est plus simple et ne nécessite pas autant de calculs, ce qui le rend d'ailleurs plus rapide. Il faut néanmoins utiliser plus d'images pour apprendre un visage et obtenir la même précision qu'avec Haar. **C'est cet algorithme que nous avons choisi pour sa rapidité et son efficacité.** Voyons comment il fonctionne.

L'algorithme consiste à parcourir les pixels de l'image en examinant le voisinage local (en général 3x3). On prend la valeur du pixel central et on applique un seuillage sur le voisinage : tous les pixels du voisinage dont la valeur est inférieure à la valeur du pixel central seront mis à 1 et les autres à 0.

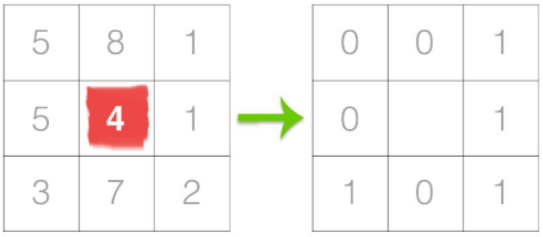



**Figure 16 : Seuillage des pixels du voisinage**

Ensuite, on calcule la valeur du pixel central de cette façon :

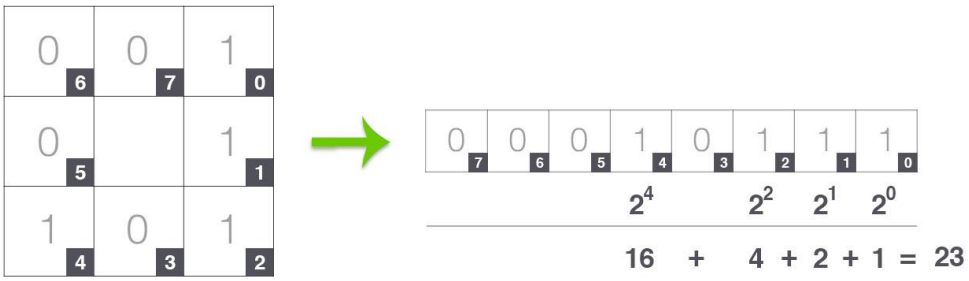



**Figure 17 : Calcul de la valeur du pixel central avec les valeurs des pixels du voisinage**

Et on remplace le pixel central par la nouvelle valeur :

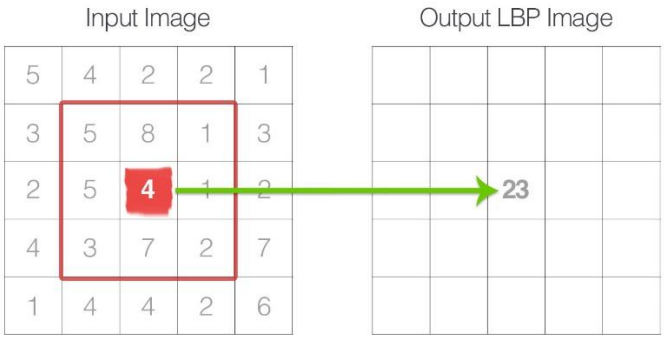



**Figure 18 : Stockage du résultat dans le pixel central**

Et voilà à quoi ressemble la représentation LBP d'une image :

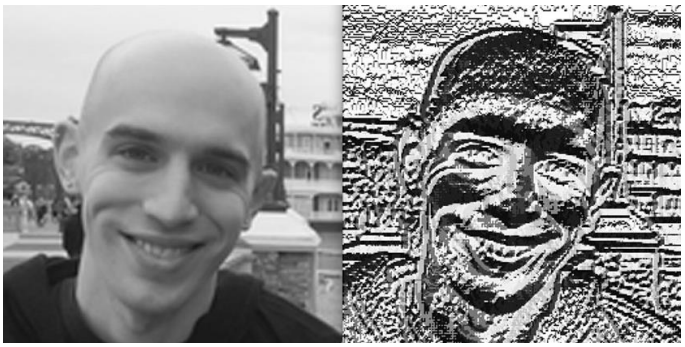


**Figure 19 : Représentation LBP d'une image**

La prochaine étape consiste à identifier les caractéristiques des visages. Il est possible d'utiliser différentes tailles de voisinage et prendre en compte plus ou moins de pixels :

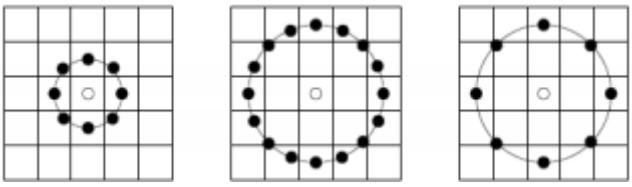


**Figure 20 : De gauche à droite : voisinage 3x3, voisinage 5x5, voisinage 5x5 avec interpolation**

La dernière étape est d'identifier les caractéristiques, en voici certaines avec des exemples de valeurs :

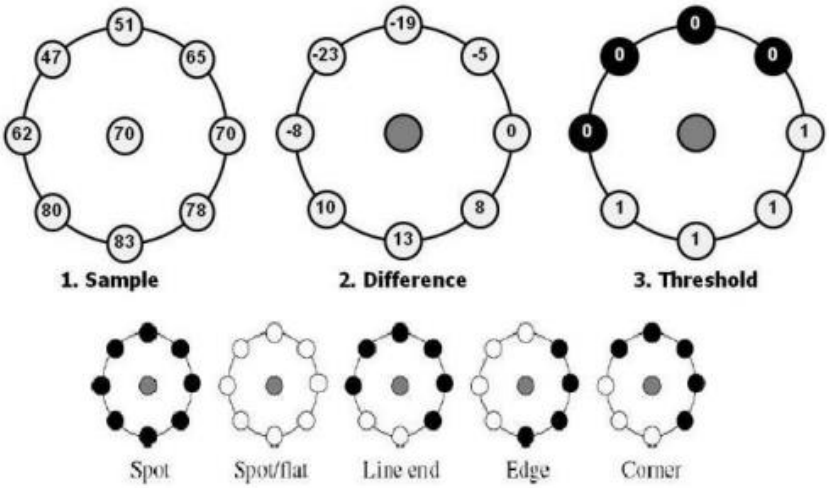



**Figure 21 : Caractéristiques LBP au voisinage 3x3**

### c) Les cascades OpenCV

Nous avons découvert les grandes lignes du fonctionnement des classificateurs Haar et LBP, il est temps de rentrer dans le vif du sujet et de comprendre ce que signifie le terme « cascade » au sein de la bibliothèque.

Une cascade est le résultat de plusieurs classificateurs simples (appelés « stages » en anglais, ou étapes). Ils sont appliqués successivement jusqu'à ce que l'un des « stages » échoue ou qu'ils soient tous validés. L'idée est de rejeter les zones ne contenant pas de visage avec le moins de calculs possibles. Le premier classificateur est donc le plus optimisé et permet de rejeter rapidement une zone si l'objet recherché ne s'y trouve pas. Si potentiellement l'objet s'y trouve, alors le deuxième classificateur est utilisé et ainsi de suite jusqu'au dernier.

C'est ce qu'on peut voir sur les photos d'Angelina Jolie (voir les encadrés rouges) :

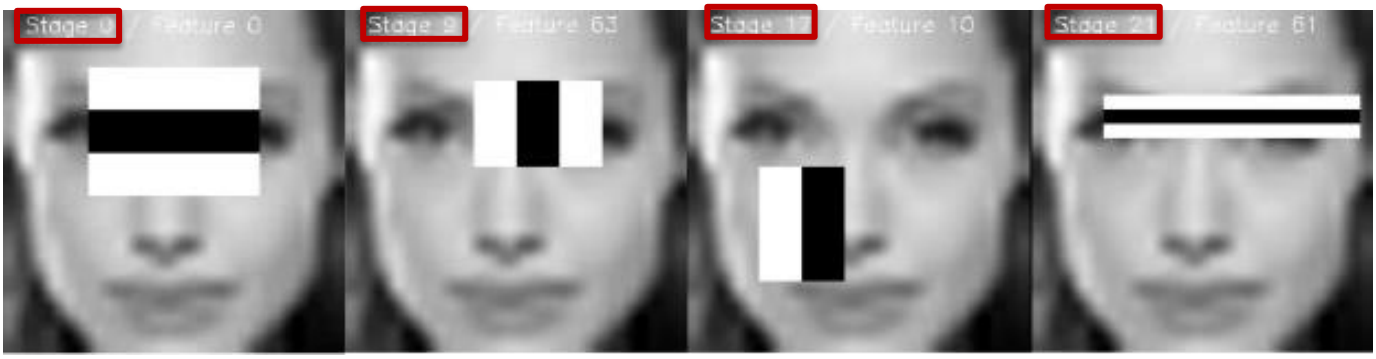



**Figure 22 : Les différentes étapes de validation de la cascade**

Revenons à notre programme. Nous avons créé la classe *AppWindow* qui hérite de la classe *Window* et qui gère la fenêtre de l'application ainsi que la détection de visages. On y a rajouté deux attributs :

```
std::vector<cv::Rect> m_faces;
cv::CascadeClassifier m_faceCascade;
```

- *m_faces* est un tableau dynamique (un *vector de la bibliothèque C++ standard*) qui stockera les boîtes englobantes sous forme de *Rect* (faisant partie d'OpenCV)
- *m_faceCascade* est la cascade de classificateurs qui nous permettra de trouver tous les visages de l'image.

Il faut maintenant charger une cascade fournie avec la bibliothèque OpenCV à l'aide de la méthode *load* :

```
m_faceCascade.load("lbpcascades\\lbpcascade_frontalface.xml");
```

On peut également charger une cascade Haar fournie également avec OpenCV :

```
m_faceCascade.load("haarcascades\\haarcascade_frontalface_default.xml");
```

Il faut bien sûr vérifier si la cascade a bien été chargée : la gestion d'erreurs est disponible dans le fichier *app_window.cpp*. La méthode load retourne *false* si la cascade n'a pas été trouvée mais peut aussi générer une exception si le format de la cascade est incorrect ! Il faut donc vérifier sa valeur de retour et la mettre dans un bloc *try/catch*.

Une fois que la cascade est chargée, nous pouvons utiliser la méthode *detectMultiScale* de *CascadeClassifier* pour trouver les visages. Nous n'avons pas encore décrit la façon dont nous récupérons le flux vidéo de la webcam. Avec OpenCV c'est un jeu d'enfant :

On crée une instance de la classe *VideoCapture*. Le premier paramètre est le numéro de la webcam (utile si vous avez plusieurs webcams connectées) et le deuxième l'API de capture vidéo à utiliser. Ici, on utilise DirectShow de Microsoft, qui offre une meilleure flexibilité en termes de choix de résolution et d'autres paramètres. L'API utilisée par défaut est MSMF (Microsoft Media Foundation).

```
cv::VideoCapture capture(0, cv::CAP_DSHOW);
```

Après avoir vérifié avec la méthode *isOpened()* si tout s'est bien passé, nous sommes prêts à récupérer le flux vidéo.

Pour ce faire, il faut créer une variable qui va accueillir les images vidéo récupérées :

```
cv::Mat frame;
```

On n'a plus qu'à mettre le code suivant dans une boucle et le tour est joué :

```
capture >> frame;
cv::flip(frame, frame, 1); // on retourne l'image sur l'axe Y pour avoir un effet miroir
cv::resize(frame, m_lastVideoFrame, cv::Size(800, 600));
```

Et on en profite pour appeler la méthode *recognizeFaces* et *drawFrame* que nous avons créées et que nous décrirons plus tard :

```
recognizeFaces();
drawFrame();
```

*Remarque : il est important de créer un thread réservé à la capture vidéo afin de ne pas bloquer la boucle d'évènements de la fenêtre, ce qui aurait pour effet de la rendre « unresponsive ». Windows afficherait « {Nom de l'application} (Ne répond pas) » dans la barre de titre du programme.*

Pour créer notre thread vidéo, nous utilisons la fonction *CreateThread* de l'API Windows (aussi appelée WinAPI ou Win32 API). Il faut inclure le fichier « windows.h » pour avoir accès à l'API native de Windows.

```
m_videoThreadHandle = CreateThread(NULL, 0, staticVideoThreadStart, (void*)this, 0, NULL);
```

Si cette fonction retourne NULL, alors la création du thread a échoué.

Pour quitter le thread vidéo, on arrête la boucle décrite plus haut puis on attend la fin du thread :

```
WaitForSingleObject(m_videoThreadHandle, 2500);
```

*Remarque : WaitForSingleObject équivaut à pthread_join sous Linux. Il y a néanmoins deux différences : cette fonction ne permet pas d'obtenir la valeur de retour du thread et le deuxième paramètre est la durée pendant laquelle cette fonction va bloquer si le thread ne se termine pas. Pour obtenir la valeur retournée par un thread, on utilise la fonction GetExitCodeThread qui renvoie soit la valeur de retour du thread soit STILL_ACTIVE si le thread ne s'est pas encore terminé.*

### d) Détection de visages : code et tests !

Voici le code permettant de détecter des visages :

```
void AppWindow::recognizeFaces()
{
    cv::Mat grayImg;
    m_faces.clear();

    cv::cvtColor(m_lastVideoFrame, grayImg, cv::COLOR_BGR2GRAY);
    cv::equalizeHist(grayImg, grayImg);

    m_faceCascade.detectMultiScale(grayImg, m_faces);

    for (size_t i = 0; i < m_faces.size(); i++)
    {
        // Nous avons volontairement effacé le code qui se trouvait à la place de ce commentaire, nous le découvrirons plus tard dans la partie « reconnaissance faciale »

        cv::rectangle(m_lastVideoFrame, m_faces[i], cv::Scalar(255, 0, 0), 2);
    }
}
```

Quelques explications s'imposent…

Dans un premier temps, nous devons convertir l'image capturée en niveaux de gris (ou grayscale en anglais). C'est ce que fait la fonction *cvtColor* avec *COLOR_BGR2GRAY* en troisième paramètre.

*Remarque : les « Mat » d'OpenCV sont par défaut en BGR (Blue Green Red). Cela est probablement dû à Windows qui utilise également ce même format dans son API native. Le format n'a pas été changé pour des raisons historiques. Windows utilise toujours ce format pour le type COLORREF qui sert à stocker des couleurs avec le format 0xAABBGGRR (A correspondant à la composante Alpha).*

On effectue ensuite une égalisation de l'histogramme de l'image. Cette étape n'est pas nécessaire mais contribue à l'amélioration du contraste de l'image, ce qui a pour effet d'augmenter la précision de la détection et de la reconnaissance faciale qu'on applique par la suite.

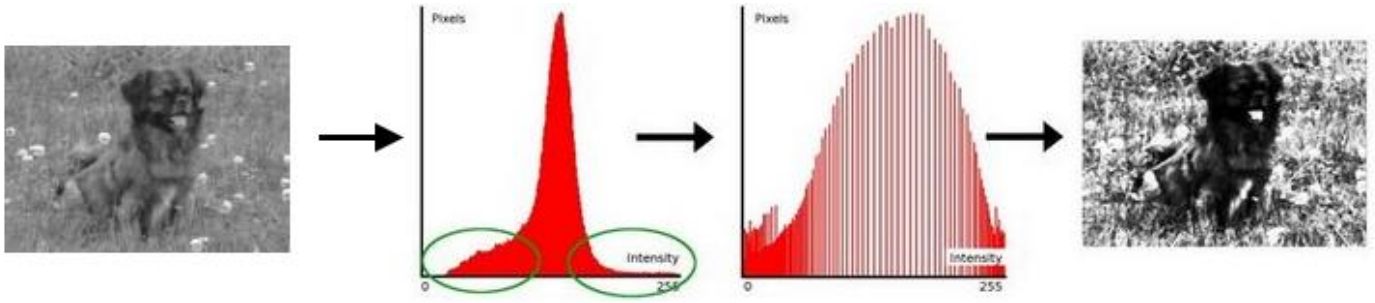



**Figure 23 : Égalisation d'histogramme avec OpenCV**

L'histogramme se voit étiré et utilise toute la gamme de couleurs disponibles au lieu d'une certaine partie restreinte.

Nous pouvons dès à présent détecter les visages de l'image avec la méthode *detectMultiScale*. Ses paramètres sont :

- *image* : l'image sur laquelle nous allons effectuer la détection.
- *objects* : tableau d'objets « Rect » qui sera rempli avec les boîtes englobantes des visages trouvés.
- *scaleFactor* : ce paramètre indique de combien sera réduite l'image à chaque redimensionnement, effectué par la fonction pour trouver des visages de toutes les tailles. La valeur par défaut est 1.1 et c'est celle que nous utiliserons par la suite. Nous avons également testé les valeurs 1.15 et 1.2 qui améliorent la vitesse d'exécution de l'algorithme mais ont pour effet de réduire le nombre de visages trouvés.
- *minNeighbors* : c'est le nombre minimal de boîtes englobantes trouvées pour un seul visage/une seule zone, permettant de valider la présence du visage. En mettant ce paramètre à 0, il y aura un taux d'erreur plus grand et beaucoup de zones ne correspondant pas à des visages seront considérées comme valides.
- *flags* : paramètre utilisé dans les anciennes versions d'OpenCV et pour les anciennes cascades.
- *minSize* : taille minimale d'une boîte englobante.
- *maxSize* : taille maximale d'une boîte englobante.

Finalement, nous utilisons une boucle pour parcourir et afficher les boîtes englobantes.

Et voici le résultat avec la cascade Haar « haarcascade_frontalface_default.xml » :

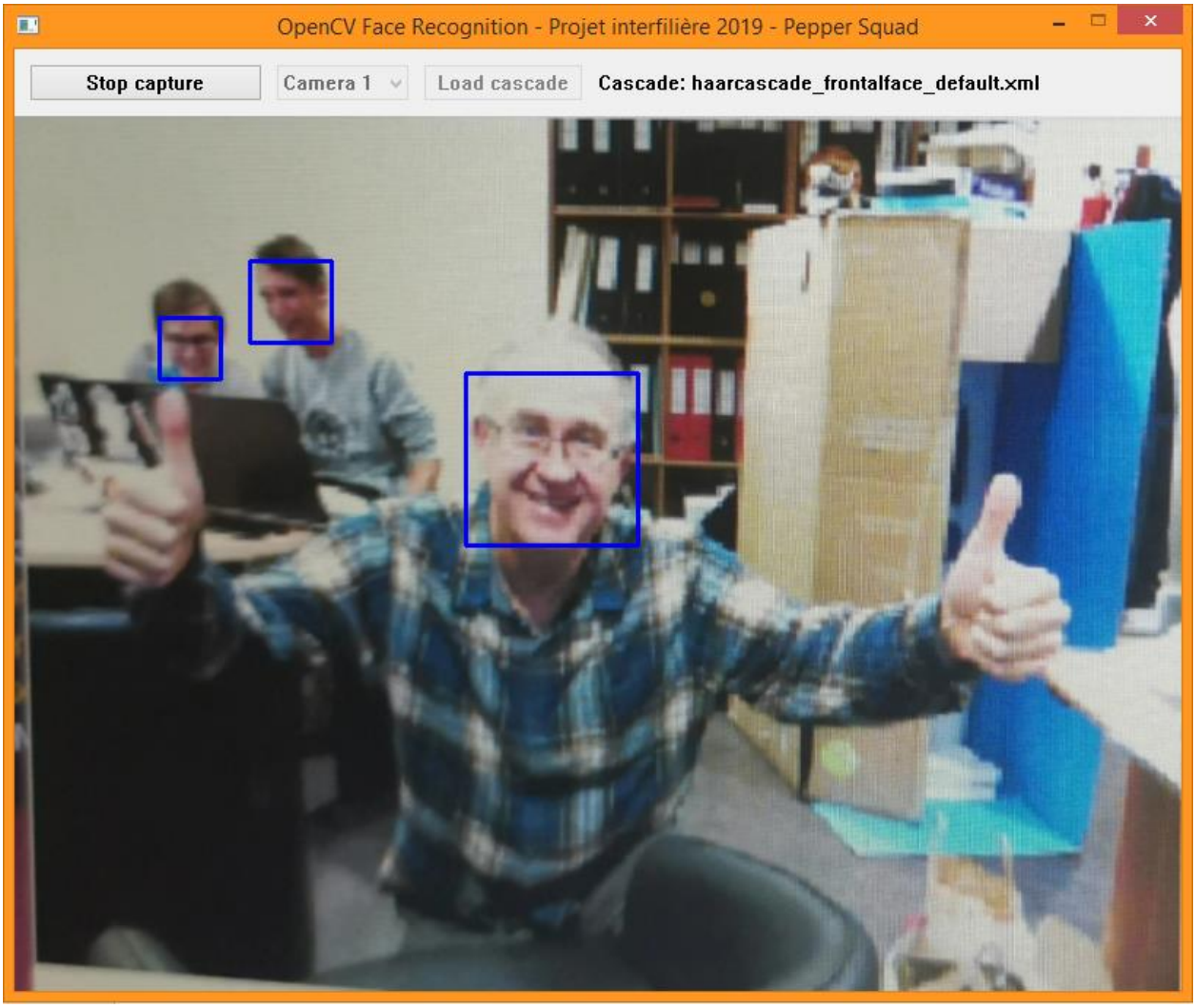



**Figure 24 : Détection de visages avec OpenCV**

On remarque que la cascade LBP « lbpcascade_frontalface.xml » ne permet pas de détecter les visages d'Axel et d'Alexandre, mais seulement le visage de Monsieur Blazevic :

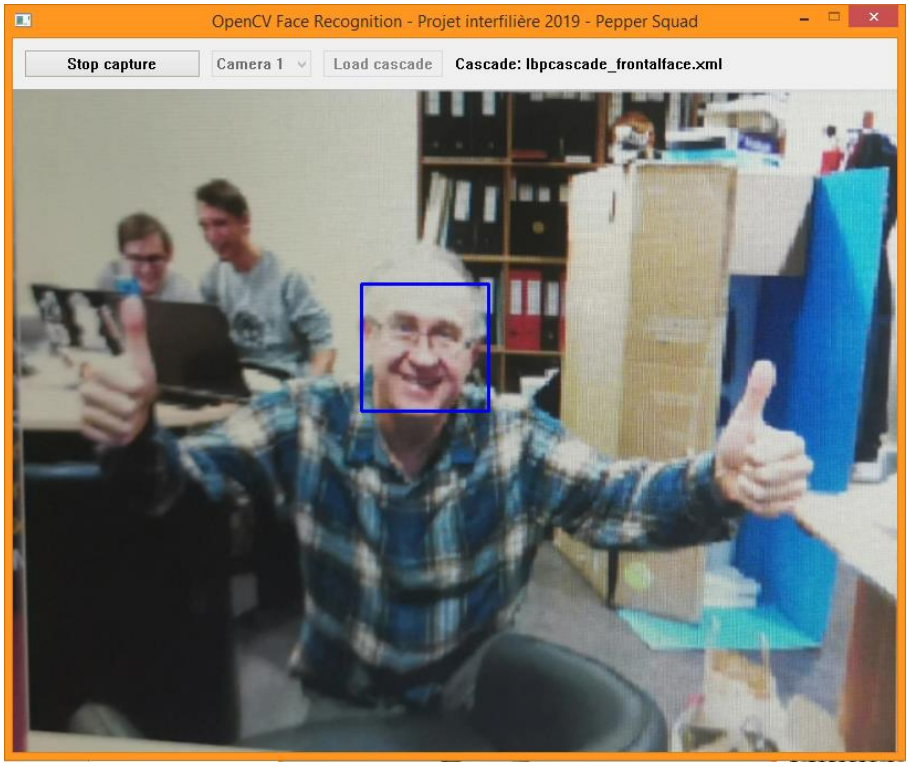



**Figure 25 : Détection de visages : deux visages détectés avec une cascade LBP**

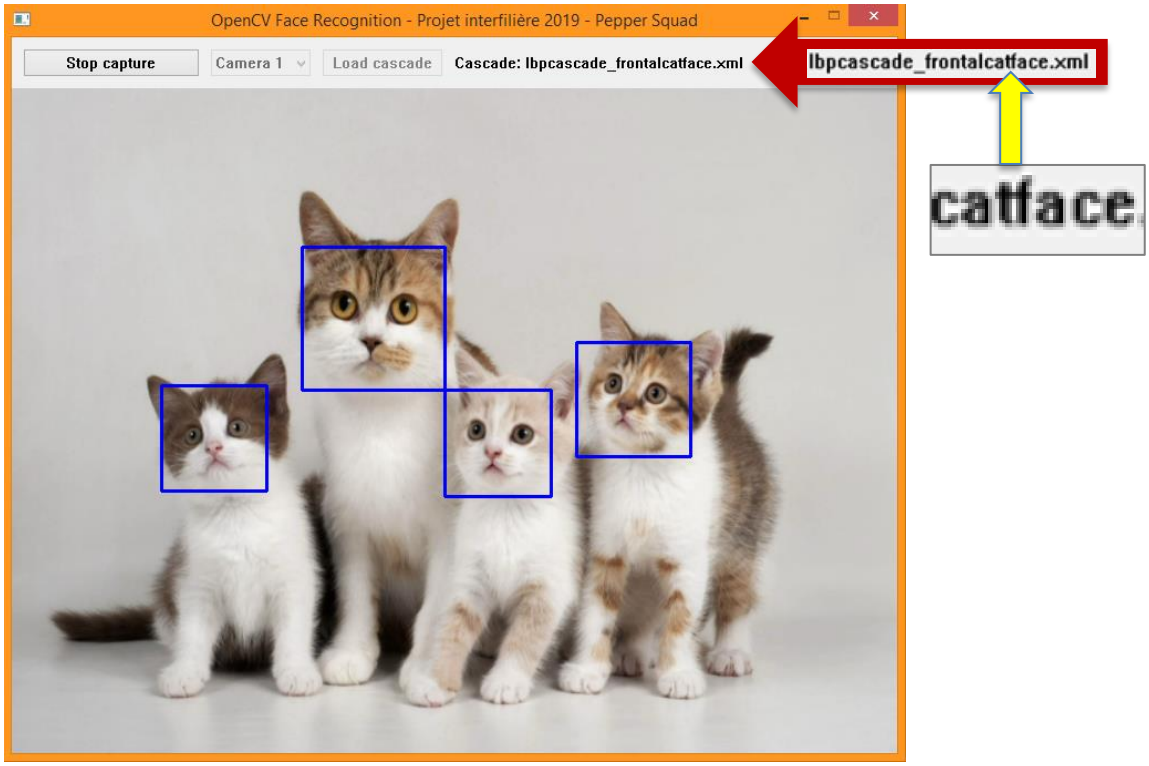



**Figure 26 : La détection à l'aide de cascades ne se limite pas seulement aux humains**

### e) Reconnaissance faciale : la théorie

Nous y voilà ! C'est la partie que nous attendions probablement tous, car même si la reconnaissance faciale avec Pepper était impressionnante et fonctionnait parfaitement, elle ne nous a pas donné autant de satisfaction que de programmer notre propre logiciel de reconnaissance faciale.

OpenCV embarque plusieurs algorithmes de reconnaissance faciale, parmi lesquels figure l'algorithme LBPH (Local Binary Patterns Histograms) qui ressemble à l'algorithme LBP pour la détection de visages, avec des étapes supplémentaires. C'est un algorithme relativement simple et qui fonctionne très bien mais qui nécessite en général une dizaine d'images de référence par personne pour commencer à obtenir des résultats intéressants.

La première étape est d'obtenir la représentation LBP de l'image source. Il faut ensuite diviser l'image en régions et extraire un histogramme de chacune d'entre elles. Les histogrammes sont ensuite concaténés pour permettre leur stockage en tant que « données d'apprentissage ». Pour reconnaître un visage, les histogrammes « appris » sont simplement comparés à l'histogramme de l'image source (provenant d'un flux vidéo par exemple). Différentes approches sont possibles pour comparer les histogrammes, la plus basique étant le calcul de la distance euclidienne.

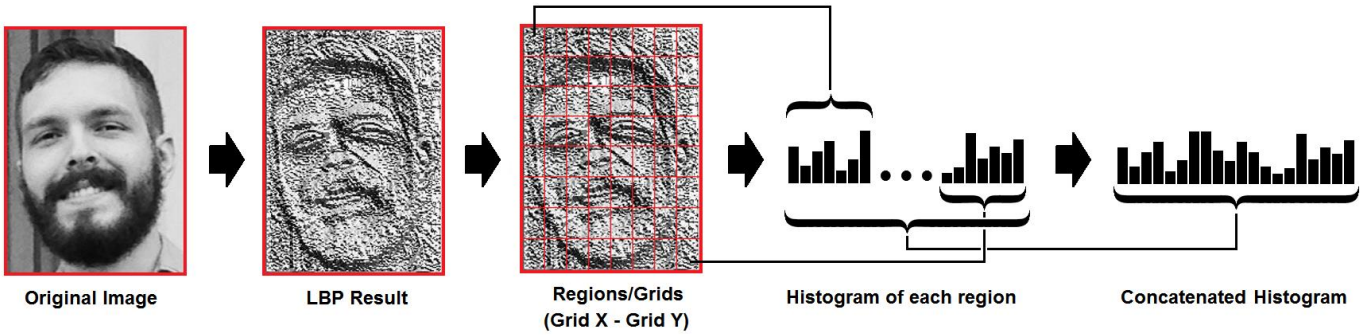



**Figure 27 : Extraction d'histogrammes à partir de régions LBP**

Voici la généralisation du théorème de Pythagore à un espace de dimension *n*, nous permettant d'obtenir la distance euclidienne entre deux histogrammes :

$$D = \sqrt{\sum_{i=1}^{n}(hist1_i - hist2_i)^2}$$

**Figure 28 : Distance euclidienne entre deux histogrammes**

### f) Reconnaissance faciale : place à la pratique !

Maintenant que nous savons comment fonctionne la reconnaissance faciale avec l'algorithme LBPH, nous allons expliquer comment nous l'avons mise en pratique avec OpenCV.

La reconnaissance faciale OpenCV se fait à l'aide de la classe *LBPHFaceRecognizer*, qui hérite de la classe *FaceRecognizer*, les deux faisant partie du namespace *cv::face*. La classe *FaceRecognizer* hérite à son tour de la classe *Algorithm* du namespace *cv*.

Nous ajoutons donc un LBPHFaceRecognizer dans les attributs de notre classe *AppWindow* :

```
cv::Ptr<cv::face::LBPHFaceRecognizer> m_faceRecognizer;
```

Il faut ensuite créer une instance de la classe *LBPHFaceRecognizer*. On utilise pour cela la méthode statique *create*, qui retourne un pointeur intelligent *cv::Ptr* sur une instance de *LBPHFaceRecognizer* créée dynamiquement. Un pointeur intelligent est un type qui permet la libération automatique de mémoire allouée dynamiquement. La mémoire peut être libérée de plusieurs façons :

- En sortie d'un bloc de code. La mémoire est libérée lorsque le pointeur intelligent qui y est associé est détruit. Un exemple d'implémentation de ce type de pointeur intelligent est *std::unique_ptr* qui fait partie de la bibliothèque C++ Standard à partir du C++11.
- Lorsqu'il n'y a plus aucune référence vers le pointeur concerné. Ce type de pointeur intelligent implémente le comptage de références. Ainsi, plusieurs pointeurs intelligents peuvent se partager un pointeur vers la même portion de mémoire, qui sera libérée une fois que tous les pointeurs intelligents auront été détruits. La bibliothèque C++ Standard implémente *std::shared_ptr* à partir du C++11.

Revenons à notre *cv::Ptr*. Il s'agit d'une implémentation propre à OpenCV du pointeur intelligent *std::shared_ptr*.

Nous créons donc un *LBPHFaceRecognizer* avec la méthode statique *create* :

```
m_faceRecognizer = cv::face::LBPHFaceRecognizer::create();
```

Si des données d'apprentissage sont disponibles dans un fichier *trained_faces.xml*, par exemple, on peut les charger avec la méthode *read* :

```
m_faceRecognizer->read("trained_faces.xml");
```

Attention, il faut mettre la ligne ci-dessus dans un bloc *try/catch* car la méthode *read* peut générer une exception si le chargement a échoué.

Il est temps d'apprendre des visages à notre *LBPHFaceRecognizer* ! Deux méthodes sont prévues à cet effet : *train* et *update*, qui ont les mêmes paramètres :

- Les images des visages à apprendre. Il s'agit d'un tableau de *cv::Mat*.
- Les « labels » ou étiquettes associées aux images. C'est un tableau d'entiers (*int*)

Chaque image passée à la fonction doit avoir une étiquette, il faut donc que les deux tableaux décrits ci-dessus aient la même longueur.

*Remarque : Plusieurs images peuvent avoir le même « label » et correspondront dans ce cas au même visage ! Cela permet d'apprendre par exemple 100 visages par personne.*

Nous passerons un *std::vector<cv::Mat>* en premier argument et les étiquettes sous forme d'un *std::vector<int>.*

La méthode *train* permet d'apprendre des visages après avoir effacé les données déjà apprises. La méthode *update*, quant à elle, permet d'apprendre de nouveaux visages sans effacer les anciens. Si aucun visage n'a encore été appris, on est libre d'utiliser l'une de ces deux méthodes.

*Remarque : la classe LBPHFaceRecognizer implémente la méthode update, mais ce n'est pas le cas des classes FisherFaceRecognizer et EigenFaceRecognizer. C'est aussi pour cette raison que nous avons choisi l'algorithme LBPH, qui supporte la méthode update ce qui le rend plus pratique d'utilisation puisqu'il permet d'apprendre de nouveaux visages sans avoir à tout réapprendre.*

Nous avons donc créé une méthode *learnFace* permettant d'apprendre un visage et de l'associer à un nom/prénom. Pour apprendre un nouveau visage avec notre logiciel, l'utilisateur doit mettre en pause le flux vidéo et cliquer sur la boîte englobante du visage à apprendre. Il doit ensuite entrer son prénom, son nom ou les deux dans le champ prévu à cet effet puis valider. Il peut ensuite relancer la vidéo et verra son visage identifié par le programme.

Voici le code de la méthode *learnFace* avec des commentaires expliquant son fonctionnement :

```
void AppWindow::learnFace(std::string name)
{
    // On met la première lettre du nom en majuscule. La longueur de la chaîne de caractères est vérifiée en amont dans notre programme donc nous pouvons accéder à la première lettre sans inquiétude.
    name[0] = std::toupper(name[0]);

   // m_hoveredFace est un cv::Rect et contient la boîte englobante du visage survolé par la souris de l'utilisateur au moment du clic. Ici, on copie la ROI (Region Of Interest) dans la variable face.
    cv::Mat face(m_lastVideoFrame, m_hoveredFace);
    // On convertit l'image en niveaux de gris
    cv::cvtColor(face, face, cv::COLOR_BGR2GRAY);
   // Et on la redimensionne en 100x100. C'est une taille relativement petite mais qui nous permet d'arriver à de bons résultats sans « perdre » trop d'informations du visage.
    cv::resize(face, face, cv::Size(100, 100));

    // La future étiquette de l'image à apprendre.
    int label;

    // On récupère les étiquettes liées au prénom de la personne.
    std::vector<int> m_foundLabels = m_faceRecognizer->getLabelsByString(name);

    // Si le nom est connu (le visage de la même personne a déjà été enregistré), alors on reprend la même étiquette pour le nouveau visage. Sinon, on essaye de trouver une étiquette « libre » puisque le visage n'est pas encore connu.
    if (!m_foundLabels.empty()) {
        label = m_foundLabels[0];
    } else {
        label = 0;

        std::vector<int> allLabels = m_faceRecognizer->getLabels();
        std::vector<int>::iterator it;
```

```
        it = std::find(allLabels.begin(), allLabels.end(), label);

        while (it != allLabels.end())
            it = std::find(allLabels.begin(), allLabels.end(), ++label);
    }

    // Le tableau qui contiendra le label du visage. Même s'il n'y a qu'un visage, nous sommes obligés d'utiliser un tableau car c'est ce que demande la fonction update.
    std::vector<int> labels;
    labels.push_back(label);

    // Le tableau qui contiendra l'image avec le visage à apprendre. Même remarque que pour les labels, il faut utiliser un tableau pour une seule image.
    std::vector<cv::Mat> images;
    images.push_back(face);

    // Et on apprend le nouveau visage sans effacer ceux déjà appris !
    m_faceRecognizer->update(images, labels);
    // Puis on associe une chaîne de caractères au label. Comme dit précédemment, il peut s'agir du prénom de la personne à qui appartient le visage.
    m_faceRecognizer->setLabelInfo(label, name);

    // Une variable indiquant si le FaceRecognizer contient des données d'apprentissage ou pas.
    m_isFaceRecognizerTrained = true;
    // Et on enregistre les données d'apprentissage dans un fichier XML ! TRAINING_DATA_FILENAME est une constante de préprocesseur que nous avons définie et qui contient le nom du fichier, à savoir « trained_faces.xml ».
    m_faceRecognizer->save(TRAINING_DATA_FILENAME);
}
```

Le code ci-dessus montre la puissance d'OpenCV : nous avons réussi à apprendre un visage en à peine quelques lignes de code !

Nous avons utilisé la méthode *save* pour enregistrer les données apprises sur le disque dur. Cette étape est bien entendu optionnelle mais si elle est omise, les données apprises seront perdues à la fermeture du programme.

La classe *LBPHFaceRecognizer* ne possède malheureusement pas de méthode pour supprimer un visage appris. Deux solutions sont possibles :

- Réapprendre tous les visages. Cette méthode est envisageable s'il n'y a pas beaucoup de visages appris et encore…
- « Parser » le fichier XML contenant les données apprises et supprimer celles de notre choix. Nous n'avons pas essayé cette technique, mais parser du XML n'est pas ce qu'il y a de plus simple et il faudrait savoir précisément comment OpenCV stocke ses données pour ne pas corrompre le fichier.

Voici le format des données d'apprentissage :

```
<?xml version="1.0"?>
<opencv_storage>
<opencv_lbphfaces>
  <!-- Paramètres de l'algorithme -->
  <threshold>1.7976931348623157e+308</threshold>
  <radius>1</radius>
  <neighbors>8</neighbors>
  <grid_x>8</grid_x>
  <grid_y>8</grid_y>
  <histograms>
    <_ type_id="opencv-matrix">
      <rows>1</rows>
      <cols>16384</cols>
      <dt>f</dt>
      <data>
  <!-- Histogramme de la première image -->
        </data></_>
  <!-- Autres images ici -->
  </histograms>
  <labels type_id="opencv-matrix">
    <rows>3</rows>
    <cols>1</cols>
    <dt>i</dt>
    <!-- Ordre d'apparition des histogrammes -->
    <data>0 1 2</data></labels>
  <labelsInfo>
    <_>
      <label>0</label>
      <value>"M. Blazevic"</value></_>
    <_>
      <label>1</label>
      <value>Axel</value></_>
    <_>
      <label>2</label>
      <value>Alex</value></_></labelsInfo></opencv_lbphfaces>
</opencv_storage>
```

Pour supprimer un visage, il faudrait donc supprimer ses histogrammes, ses labels ainsi que ses informations additionnelles et modifier la valeur qui se trouve entre les balises `<rows></rows>`.

Et voici le code que nous utilisons pour reconnaître des visages :

```
int label; // Le label du visage reconnu.
double confidence; // Valeur de confiance, ou plutôt de distance, calculée par l'algorithme LBPH. C'est la distance entre l'histogramme de référence et celui de l'image source. Le visage reconnu est celui avec la plus petite distance calculée. Pour plus de détails, voir la partie théorique sur LBPH.
cv::Mat face(grayImg, m_faces[i]); // On copie la ROI
m_faceRecognizer->predict(face, label, confidence); // Et on reconnaît le visage !
```

La reconnaissance faciale se fait avec la méthode *predict*, qui prend au plus trois paramètres :

- L'image source. L'avantage avec LBPH, c'est que la taille de cette image peut être différente de celle de l'image de référence. Avec FisherFaces et EigenFaces, il faut qu'elles soient de la même taille.
- Le label du visage reconnu. Cette variable est passée par référence. De manière générale, OpenCV utilise beaucoup les références dans ses fonctions, ainsi que la surcharge.
- La distance, pouvant faire office de valeur de confiance.

Maintenant que nous savons faire de la détection et de la reconnaissance de visages, voici le code complet de notre méthode *recognizeFaces* :

```
void AppWindow::recognizeFaces()
{
    cv::Mat grayImg;
    m_faces.clear();

    cv::cvtColor(m_lastVideoFrame, grayImg, cv::COLOR_BGR2GRAY);
    cv::equalizeHist(grayImg, grayImg);

    m_faceCascade.detectMultiScale(grayImg, m_faces); // Détection des visages

    for (size_t i = 0; i < m_faces.size(); i++)
    {
        if (m_isFaceRecognizerTrained) {
            int label;
            double confidence;
            cv::Mat face(grayImg, m_faces[i]);
            m_faceRecognizer->predict(face, label, confidence); // Reconnaissance faciale

            int captionHeight = 16;
            cv::Rect captionRect(m_faces[i].x - 1, m_faces[i].y - captionHeight, m_faces[i].width + 2, captionHeight);
            // On affiche un rectangle bleu sur lequel on écrira le nom de la personne
            cv::rectangle(m_lastVideoFrame, captionRect, cv::Scalar(255, 0, 0), -1);

            int baseline = 0;
            cv::String caption = std::to_string(label) + ": " + m_faceRecognizer->getLabelInfo(label) + " (" + std::to_string((int)confidence) + ")";
            cv::Size textSize = cv::getTextSize(caption, cv::FONT_HERSHEY_PLAIN, 1.0, 1, &baseline);
            cv::Point textPosition(captionRect.x + (captionRect.width - textSize.width) / 2, captionRect.y + captionRect.height - 1);
            // Affichage du nom de la personne au dessus de la boite englobante
            cv::putText(m_lastVideoFrame, caption, textPosition, cv::FONT_HERSHEY_PLAIN, 1.0, cv::Scalar(255, 255, 255), 1);
        }

        // Affichage d'un rectangle bleu représentant une boite englobante
        cv::rectangle(m_lastVideoFrame, m_faces[i], cv::Scalar(255, 0, 0), 2);
    }
}
```

**Figure 29 : Code de la méthode recognizeFaces**

Cette méthode détecte les visages puis les parcourt un par un, après quoi elle leur applique la méthode *predict* et dessine les boîtes englobantes ainsi que les noms des personnes.

Et voilà le résultat :

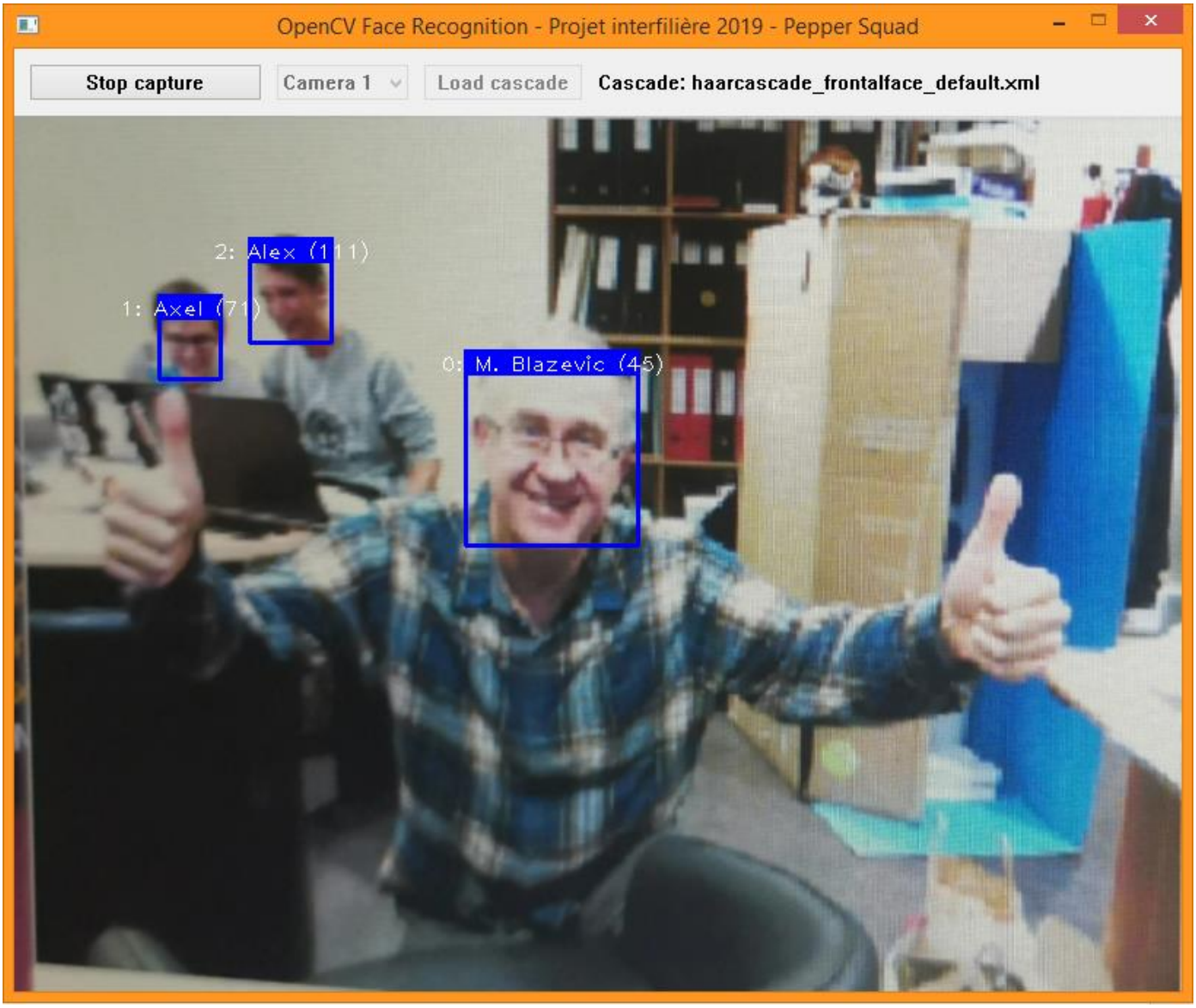



**Figure 30 : Reconnaissance faciale avec OpenCV**

# VIII. Utilisation des Landmarks

En continuant notre exploration dans le module « Vision » fournit dans le SDK par SoftBankRobotics nous avons trouvé un service intéressant : `'ALLandMarkDetection'`, qui est un service qui permet au robot de reconnaître un marqueur spécial qui a un pattern spécifique qui s'appelle « Naomarks ».

## 1. Explication

Ce « Naomark » est composé d'un cercle noir avec un motif blanc. L'encodage sous la forme du motif blanc est l'identifiant unique du naomark. SoftBankRobotics nous fournit gratuitement 30 Naomarks.

Le service de détection de points de repère permet d'acquérir les informations de ces marqueurs lorsqu'une des caméras de Pepper en détecte un.

Ces informations sont l'identifiant du marqueur détecté (qu'il détecte grâce au motif blanc), la distance du naomark par rapport à la caméra du robot et la forme des naomarks.

Les naomarks ressemblent à cela :

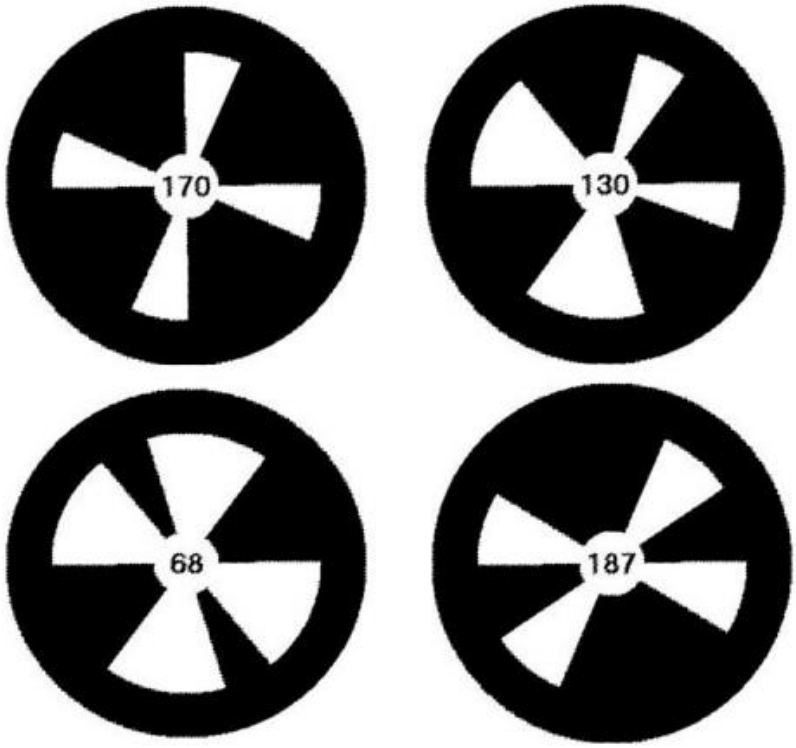



**Figure 31 : Exemples de Naomarks (une sorte de marqueur)**

## 2. Cas d'utilisation

Nous avons donc décidé d'utiliser ces landmarks pour générer des actions quand le robot en voit un, ou lui permettre d'acquérir des informations sur la pièce où se situe le landmark.

Dans un premier temps, regardons ce que nous renvoie l'événement `"LandmarkDetected"` :

```
ALLandMarkDetection {
  TimeStamp,
  MarkInfo[N],
  CameraPoseInFrameTorso,
  CameraPoseInFrameRobot,
  CurrentCameraName
}
```

- **TimeStamp** qui est une structure contenant un champ « secondes », et « microsecondes », correspond à l'horodatage de l'image qui a été utilisé pour effectuer la détection.
- **MarkInfo** il y a autant de MarkInfo que de naomark détecté.

- **CameraPoseInFrameTorso** : *Position6D* de la caméra au moment de la prise de vue, dans *FRAME_TORSO*.
- **CameraPoseInFrameRobot** : *Position6D* de la caméra au moment de la prise de vue, dans *FRAME_ROBOT*.
- **CurrentCameraName** : Nom de la caméra utilisée pour détecter le naomark (« CameraTop » ou « CameraBottom »).

Nous allons voir en détail ce que comporte le tableau **Markinfo** :

```
MarkInfo {
  ShapeInfo,
  MarkID
}
```

- **MarkID** qui est un champ de type int, c'est l'identifiant du naomark.
- **ShapeInfo** est une structure qui contient :

```
ShapeInfo {
  1,
  alpha,
  beta,
  sizeX,
  sizeY,
  heading
}
```

  - **alpha** et **beta** qui représente l'emplacement du centre du naomark en termes d'angles de caméra en radian.
  - **sizeX** et **sizeY** qui sont la taille du naomark par rapport aux angles de la caméra.
  - **heading** correspond à l'orientation du naomark autour de l'axe vertical par rapport à la tête du robot.

Maintenant que nous savons ce que nous renvoie cette fonction, on peut alors imaginer plusieurs cas d'utilisation.

Tout d'abord voici un aperçu de ce que voit le robot quand un ou plusieurs naomarks sont détectés :

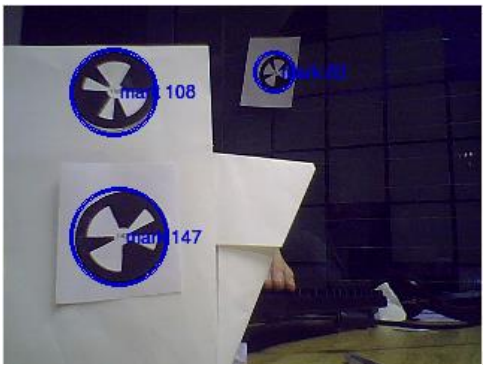



**Figure 32 : Vision du robot quand il détecte un naomark**

Le premier cas d'utilisation qu'on a imaginé, est qu'à chaque fois qu'un naomark est détecté, en fonction de l'identifiant, le robot fait une action ou affiche des informations sur la tablette.

Ceci est facilement programmable avec un switch ou une comparaison sur le champ **MarkID** du tableau **MarkInfo**.

Par exemple, si le robot détecte un naomark qui a pour identifiant 68, alors le robot affiche le plan des salles du 3ème étage de l'ISTY Mantes, et si le robot détecte un naomark qui a pour identifiant 114 alors la tablette affiche l'application « Cafet' » qui permet de commander une boisson ou un encas à la cafétéria du BDE au 2ème étage.

Exemple de code qui permet de réaliser l'exemple ci-dessus :

```
pepper.ALMemory.subscriber('LandmarkDetected').then(function(subscriber) {
    subscriber.signal.connect(function(info) {
        if(info[1].length === 0)
            return;

        switch(info[1][0][1]) {
            case '68':
                pepper.ALTabletService.showWebview('http://www.isty.uvsq.fr/'); //Affiche le siteweb de l'ISTY
                break;
            case '114':
                // on affiche l'application de la cafetaria ...
                break;
        }
    });
});
pepper.ALBarcodeReader.subscribe('LandmarkDetected');
```

Dans un premier temps, on souscrit l'événement `'LandmarkDetected'` puis on attache un gestionnaire d'évènements via ALMemory comme décrit ci-dessus, nous faisons une première vérification sur la taille du tableau grâce à l'opérateur « === ».

Le **MarkInfo** étant à l'index **1** du tableau "info", on prend le premier élément ce qui donne info[1][0] puis on va chercher le **MarkId** à l'index **1** du MarkInfo, donc on a **info[1][0][1]**.

Puis on regarde si l'identifiant correspond à un des choix du switch alors le robot fait l'action qui correspond.

Par exemple pour le naomark 68, le site de l'ISTY sera affiché sur la tablette dès la détection du naomark.

Le deuxième cas d'utilisation est un peu plus complexe. En effet, grâce aux informations du tableau **ShapeInfo**, on peut effectuer une localisation visuelle du robot par rapport au naomark.

Si on désigne la variable *S* qui est la distance du naomark par rapport à la caméra du robot, on peut la calculer en utilisant la dimension physique du naomark et les informations contenues dans le tableau **ShapeInfo**.

Notons *m* la dimension physique du naomark et *a* la taille angulaire :

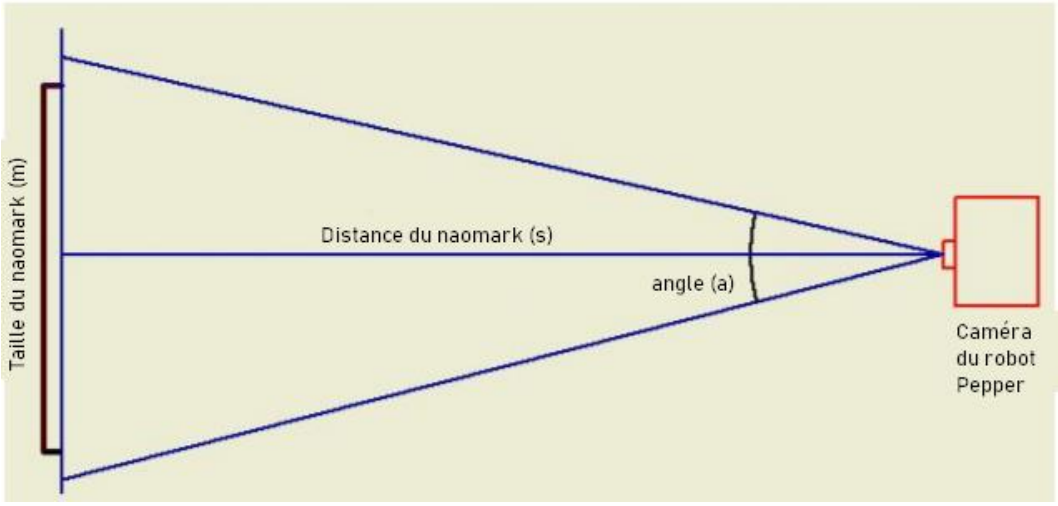



**Figure 33 : Schéma naomark - Pepper**

On obtient alors cette formule : $S = \frac{\frac{m}{2}}{\text{atan}\left(\frac{a}{2}\right)}$

Il faut ensuite utiliser les angles alpha et beta pour effectuer une transformation. Le résultat de cette transformation est une matrice qui inclut les coordonnées (x,y,z) du naomark dans le repère du robot.

L'emplacement du robot par rapport au naomark est alors déterminé en fonction des coordonnées du naomark dans le repère du robot et de ces mêmes coordonnées dans le repère global.

Landmark

$$\begin{pmatrix} X1_{global} \\ Y1_{global} \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\varphi & -\sin\varphi & X_0 \\ \sin\varphi & \cos\varphi & Y_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X1_{robot} \\ Y1_{robot} \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} X2_{global} \\ Y2_{global} \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\varphi & -\sin\varphi & X_0 \\ \sin\varphi & \cos\varphi & Y_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X2_{robot} \\ Y2_{robot} \\ 1 \end{pmatrix}$$

Pour localiser le robot, il faut utiliser deux naomark et résoudre ces 4 équations à 3 inconnues :

$X1_{global}$ $Y1_{global}$ et $X2_{global}$ $Y2_{global}$ sont les coordonnées des naomarks dans le repère global, ceux qui ont pour indice *robot* sont les coordonnées des naomarks dans le repère du robot.

X0 et Y0 sont les coordonnées du robot Pepper dans le repère global et l'angle $\varphi$ correspond à l'orientation du robot dans le repère global.

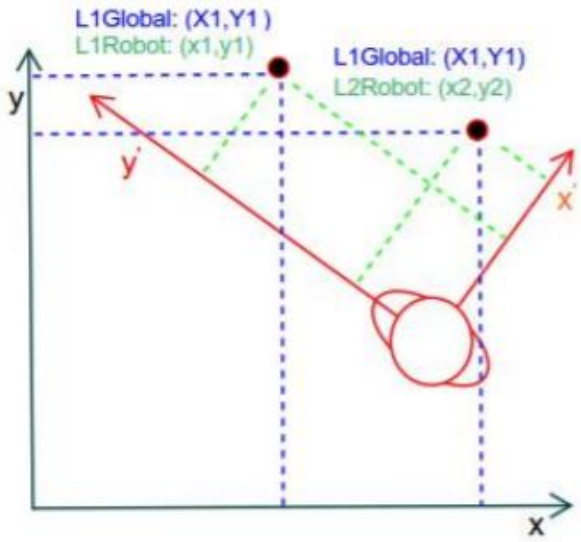



**Figure 35 : Postions des naomarks dans le repère global et le repère du robot**

Cette localisation nécessite d'autres contraintes comme :

- Le fait que les naomarks ne doivent pas être coplanaires sinon on aurait des erreurs d'estimation importantes.
- La portée de détection est limitée par le champ de vision de la caméra et la plage d'inclinaison du naomark.
- Il ne faut pas que le robot ait une grande proximité (< 30 cm) des repères sinon les estimations de postions seront incohérentes.

Nous avons pu tester que le premier cas d'utilisation, c'est-à-dire la détection d'un landmark et la reconnaissance de l'identifiant de ce landmark.

## 3. Limites

Bien que le module de détection de landmarks offre une approche simple de la localisation visuelle, il existe plusieurs limites à son utilisation. Ce module est en proie à la qualité des images acquises par la caméra frontale du robot Pepper.

Comme précisé dans la documentation, le premier point d'exigence est d'avoir un éclairage suffisant. En effet la détection se repose sur des différences de contraste de l'image. L'éclairage nécessaire est de 100 à 500 lux, un éclairage inférieur à cette plage peut entrainer des erreurs d'identifications des naomarks voir aucune détection.

La deuxième limitation concerne l'inclinaison du plan du marqueur par rapport à la caméra, qui doit être comprise entre +/- 60 degrés de détectabilité.

Pour des performances optimales, il faut donc que le naomark soit posé sur un mur et positionné à la hauteur de la caméra frontale de Pepper.

## 4. Alternative

Nous avons pensé à une alternative aux landmarks ou qui pourrait être utilisée en complémentarité avec les landmarks, et qui correspond plus à notre scénario, c'est l'utilisation de QR code.

En effet, nous avons remarqué que dans le module vision il y avait un service nommé `'ALBarcodeReader'` qui permet de gérer la détection de QR code par le robot Pepper.

Les limites d'utilisations sont pratiquement les mêmes que les landmarks. La résolution de l'image doit être réglée en fonction de la distance entre la caméra et le code lui-même, et le taux de détection dépend de la qualité et de la taille du code imprimé.

Mais son utilisation est plus simple, puisqu'on peut utiliser un générateur de QR codes et donc ne pas être limité par le nombre de landmarks disponibles, mais aussi parce que le tableau de données ne comporte que deux champs :

```
CodeData = [
  Data,
  Position
]
```

- **Data** est un champ de type string qui contient les données qui sont dans le QR code.
- **Position** qui est un tableau contenant les coordonnées des quatre coins du QR code.

On peut alors aisément écrire une condition dans notre code ou un switch avec un *strcmp* qui en fonction de chaque QR code détecté, le robot fait une action prédéfinie ou affiche des informations complémentaires sur la tablette.

On peut imaginer dans le cas de notre scénario, mettre des étiquettes avec un QR code dans chaque pièce de l'ISTY, et à chaque fois que le robot Pepper lit un QR code (ou un naomark) alors le robot se met à afficher l'emploi du temps de la salle ou des informations complémentaires.

Si le QR code se situe à l'étage, Pepper pourrait afficher le plan de l'étage par exemple.

Malheureusement, nous n'avons pas pu tester cette méthode sur le robot Pepper suite au coronavirus mais nous avons quand même pu programmer une utilisation de ce service :

```
pepper.ALMemory.subscriber('BarcodeReader/BarcodeDetected').then(function(subscriber) {
    subscriber.signal.connect(function(info) {
        if(info[0].length === 0)
            return;

        switch(info[0][0]) {
            case 'SITEISTY':
                pepper.ALTabletService.showWebview('http://www.isty.uvsq.fr/'); //Affiche le siteweb de l'ISTY
                break;
            case 'PLAN2EME':
                // on affiche le plan du 2ème étage ...
                break;
            case 'CAFET':
                // on affiche l'application de la cafetaria ...
                break;
        }
    });
});
pepper.ALBarcodeReader.subscribe('BarcodeReader/BarcodeDetected');
```

Pour détecter le QR code nous devons souscrire à l'événement `'BarcodeReader/BarcodeDetected'` puis on attache un gestionnaire d'évènements via ALMemory comme décrit ci-dessus, nous faisons une première vérification sur la taille du tableau grâce à l'opérateur « === » qui existe en JavaScript (et non en C/C++) qui permet une égalité stricte, l'opérateur d'égalité stricte renvoie true si les opérandes sont strictement égaux, aucune conversion de type n'est effectuée.

Une fois qu'on est sûre que Pepper ait détecté un QR code, on effectue un switch sur le contenu « data » qui est dans le QR code scanné.

Par exemple, ce QR code comporte comme donnée : SITEISTY (vous pouvez le vérifier avec un QR code Reader)

**Figure 36 : QR code (contenant "SITEISTY" dans data)**

Quand Pepper lira ce QR code alors d'après la condition dans le switch, il affichera le site-web de l'ISTY sur la tablette.

On peut imaginer d'autres utilisations comme dit précédemment, afficher le plan d'un étage ou une application dédiée.

# IX. Objectifs non atteints à cause du Covid-19

Lors de la première phase du projet nous avons mis en place un tableau d'objectifs à réaliser qui permettait d'atteindre l'objectif principal qui était d'utiliser Pepper pour effectuer une visite de l'école.

Voici le tableau avec l'état de la réalisation de ces objectifs :

| ***Missions à effectuer*** | ***Réalisée*** |
|---|---|
| Créer une application .apk sous Unity permettant de faire fonctionner Pepper en utilisant des blocs | Non |
| Capitaliser l'ensemble des documents sur Pepper, et créer des procédures | Oui |
| Contrôler Pepper avec une manette à Joystick | Oui |
| Utiliser le retour d'image et la reconnaissance faciale de Pepper afin de reconnaître les personnes en face de Pepper | Oui |
| Utiliser les capteurs de positionnement de Pepper afin d'éviter des obstacles lors de ses déplacements | Non |
| Actionner les bras et la tête de Pepper lorsqu'il communique ou pour serrer la main d'un individu | Non |
| Affecter des réponses types à Pepper lorsqu'on communique avec lui afin de pouvoir tenir une communication avec lui | Non |
| Effectuer une vidéo de présentation de l'ensemble des fonctionnalités que nous avons créées pour Pepper | Non |
| Implémenter une architecture ROS et utiliser OpenCV sur Pepper | Oui |
| Simuler l'ensemble des fonctions sous Gazebo et effectuer une comparaison entre la simulation et la réalité | Non |
| Afficher des animations et/ou une web App sur la tablette de Pepper | Oui |

Malheureusement, suite à un évènement exceptionnel, nous n'avons pas pu réaliser tous ces objectifs puisque les heures du projet inter-filières n'ont pas pu être effectuées entièrement et suite au confinement nous n'avons pas eu accès au robot Pepper.

Nous avons quand même pu réaliser certains objectifs :

- Capitaliser l'ensemble des documents sur Pepper et créer des procédures.
- Contrôler Pepper avec une manette à Joystick (en JavaScript).
- Utiliser le retour d'image et la reconnaissance faciale de Pepper et saluer la personne (en JavaScript).
- Implémenter une architecture ROS et utiliser d'OpenCV sans le robot Pepper (en C++).
- Afficher des sites web et vidéos sur la tablette de Pepper (en JavaScript).
- Utiliser les haut-parleurs de Pepper pour reproduire un signal audio sinusoïdal à une certaine fréquence.
- Simuler le robot Pepper sous Gazebo sans effectuer la comparaison avec le robot réel.
- Utiliser la reconnaissance des « Landmarks » (en JavaScript).

**Figure 37 : Pepper aidant au déconfinement à Tokyo**

**Figure 38 : Pepper avec un masque**

# X. Conclusion

Nous avons réussi les principales missions qui nous ont été confiées, à savoir prendre en main le robot Pepper.

En effet, nous avons réussi à le faire mouvoir avec notre propre programme écrit en JavaScript, ainsi qu'utiliser les fonctions multimédia proposées par le robot Pepper (telles que la tablette ou les haut-parleurs).

Malgré les circonstances exceptionnelles, ce projet nous a permis de mettre en pratique les notions théoriques vues en cours, ainsi que la mise en commun des compétences acquises dans chaque filière (Mécatronique, SEE, IATIC).

Ce projet nous a également fait découvrir de nouveaux outils comme le SDK NAOqi et OpenCV, ainsi que des notions de robotique et de vision pour certaines promotions.

# XI. Annexe

***INSTALLATION DE ROS INDIGO / UBUNTU :***

Nous avons créé un tutoriel pour simplifier l'installation de la version ROS Indigo sur Ubuntu 14.6, si vous voulez le consulter vous pouvez le retrouver sur notre Github : https://github.com/LexaHoods/PepperProjetInterfiliere

Vous retrouverez aussi nos différents programmes fonctionnels sur le robot Pepper.

ACRONYMES :

**SDK :** Soft Development Kit, kit de développement fournit par le fabriquant.

**API :** Application Programming Interface, qui est une interface de programmation d'application reliant le SDK et le code.

**OpenCV :** Open Computer Vision, bibliothèque open source pour la vision par ordinateur

API Windows, WinAPI, API Win32 : API native de Windows

**LBP :** Local Binary Patterns

**LBPH :** Local Binary Patterns Histograms

**CRT :** C Run-Time

**XML :** eXtensible Markup Language, c'est un langage de balisage générique

**STD Library :** C++ Standard Library, c'est la bibliothèque standard du C++

**MSVC :** Compilateur de Microsoft, fourni avec Visual Studio

**HTML :** HyperText Markup Language

**CSS :** Cascading Style Sheets

**JS :** JavaScript

**IP :** Internet Protocol

**QR Code :** Quick Response Code

**ROI :** Region Of Interest

**Naomark /landmark :** marqueur de position.

**Position6D :** Vecteur de 6 dimensions composé de 3 translations (en mètre) et 3 rotations (en radian).

**FrameTorso :** L'une des 3 références spatiales utilisées par le module ALMotion, ceci est rattaché à la référence au torse de Pepper, donc se déplace avec Pepper pendant qu'il se déplace et qu'il change d'orientation.

**FrameRobot :** L'une des 3 références spatiales utilisées par le module ALMotion, ceci est la moyenne des positions de deux pieds projetés autour d'un axe z vertical.

BIBLIOGRAPHIE :

**Aide ROS :** http://wiki.ros.org/rostopic
**SDK C++ :** https://qisdk.softbankrobotics.com/sdk/doc/pepper-sdk/index.html

**Tutoriels sur NAOqi :**

http://doc.aldebaran.com/2-5/dev/cpp/helloworld_detailed.html

http://doc.aldebaran.com/2-5/naoqi/index.html

http://doc.aldebaran.com/2-5/getting_started/creating_applications/index.html

http://doc.aldebaran.com/2-5/dev/tutos/create_a_new_service.html

**Documentation qiBuild :** http://doc.aldebaran.com/qibuild/index.html

**SDK JavaScript** : http://doc.aldebaran.com/2-5/dev/js/index.html

**Documentation NAOqi :** http://doc.aldebaran.com/2-5/naoqi/core/index.html

**GitHub OpenCV :** https://github.com/opencv

**Documentation OpenCV :**

https://docs.opencv.org/master/
https://docs.opencv.org/master/d3/d52/tutorial_windows_install.html
https://docs.opencv.org/master/d1/de5/classcv_1_1CascadeClassifier.html
https://docs.opencv.org/master/dd/d65/classcv_1_1face_1_1FaceRecognizer.html
https://docs.opencv.org/master/df/d25/classcv_1_1face_1_1LBPHFaceRecognizer.html
https://docs.opencv.org/master/dd/d7c/classcv_1_1face_1_1EigenFaceRecognizer.html
https://docs.opencv.org/master/d2/de9/classcv_1_1face_1_1FisherFaceRecognizer.html

**JavaScript :**

*Gamepad API :* https://developer.mozilla.org/en-US/docs/Web/API/Gamepad_API

*Promise :* https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

**NAOmark / Landmark / QR code:**

http://doc.aldebaran.com/2-5/naoqi/vision/index.html

http://doc.aldebaran.com/2-5/naoqi/vision/allandmarkdetection.html#allandmarkdetection

http://doc.aldebaran.com/2-5/naoqi/vision/albarcodereader.html

https://pdfs.semanticscholar.org/69ac/48384f817ae509756e85f94f1aa0137bbbaf.pdf

**Glossaire NAOqi :** http://doc.aldebaran.com/2-5/glossary.html#term-position6d

Pepper aidant pendant le confinement à Tokyo : https://www.dailysabah.com/business/tech/robots-on-the-frontlines-tokyo-quarantine-hotels-welcome-contactless-help