Rapport de projet

Interconnexion d'une plateforme de réseaux de capteurs vidéo à l'Internet avec visualisation sur un navigateur Web

ALAVAREZ Emmanuel
JIMENEZ Yvan
Master Technologies de l'Internet - 1ère année

Rensponsable : M. PHAM C. (LIUPPA)

Table des matières

1 Introduction **7**

2 Problématique **9**
2.1 Présentation 9
2.2 État de l'art 11

3 Travail réalisé **15**
3.1 Contexte 15
3.2 Site internet 15
3.3 Récupération des données et affichage 16
3.4 Communication sans fil 20

4 Conclusion **21**

A Code NesC (extraits) **25**

B Code JAVA (extraits) **27**
B.1 Récupération d'image 27
B.2 Affichage d'une image 29

Avant-propos

Ce document a pour but d'expliquer les difficultés rencontrées, les choix et décisions qui ont été nécessaires afin de parvenir à la réalisation des objectifs fixés, ceux-ci ayant changés au fur et à mesure de l'avancement du projet.

Le document n'explique que très brièvement le fonctionnement général du programme réalisé, il n'a pas pour but d'être une notice d'utilisation, mais juste de donner une idée des possibilités qu'il offre, en plus de justifier de la présence de ces fonctions.

Chapitre 1

Introduction

Un capteur est un appareil capable de mesurer des conditions physiques et/ou environnementales, telles que les sons, les vibrations, les températures, les vibrations, les mouvements... Les récentes avancées technologiques ont permis la production de capteurs ayant de faibles coûts de fabrication, très économes en énergie, de petite taille et capables de communiquer entre eux via une technologie sans fil, ce qui leur permet de travailler ensemble. De plus,ces capteurs ne se limitent pas à une seule fonction, par exemple, un capteur utilisé pour des mesures de température peut très bien être utilisé par la suite pour capturer de la vidéo.

Les possibilités de communication de tels capteurs permettraient alors de réaliser, dans le cadre de capteurs dotés de caméras, un système de vidéo-surveillance : plusieurs capteurs envoient des flux vidéo vers un capteur "central" qui lui renvoie ces données pour affichage sur un écran, par exemple. La vidéo-surveillance peut évidemment être appliquée à plusieurs domaines, allant du militaire à l'environnemental. Elle peut apporter un plus à une surveillance classique sans visualisation de l'image. Par exemple, dans le cas d'une surveillance des feux d'incendies, un capteur de température donne déjà une bonne indication, mais la possibilité d'avoir une image du lieu est un plus non négligeable, qui permettrait d'avoir des certitudes sur l'existence d'un incendie. Du fait de leur capacité à communiquer entre-eux, le capteur de température pourrait informer le capteur vidéo de la nécessite de capturer la scène.

Dans le cadre de notre TER, il nous est demandé de réaliser une application permettant la visualisation d'un flux vidéo provenant d'un capteur vidéo. Ce dernier envoie les données vers un capteur "base" connecté par liaison filaire à l' ordinateur.

Chapitre 2

Problématique

2.1 Présentation

Les capteurs que nous utilisons tournent sur l'environnement de programmation TinyOS [1] : il s'agit d'un système d'exploitation open-source spécialement conçu pour des capteurs sans-fil. Celui-ci offre une bibliothèque de composants riche, comprenant plusieurs protocoles réseaux, pilotes de capteurs et outils d'acquisition de données. Il est programmé en NesC [2], un langage de programmation basé sur le langage C, dont la propriété principale est la faible utilisation des ressources (mémoire, autonomie des batteries...).

Dans le but de réaliser ce projet, nous disposons de :

- 2 x IPR2400 [3], appelé communément Imote2 : il s'agit d'un capteur avec possibilité de lui adjoindre une extension afin de l'utiliser dans un domaine précis : il peut accepter des capteurs de température, de luminosité, d'humidité... Dans notre cas, il s'agira évidemment d'un capteur vidéo. Le laboratoire LIUPPA travaille sur ces mêmes capteurs vidéo[1].

- 1 x IMB400 [4]. Il s'agit d'une carte multimédia dotée d'une caméra qui est une extension pour l'Imote2.

- 1 x IIB2400 [5]. Carte utilisée pour le débugage des capteurs auxquels elle se connecte.

Le schéma de connexion idéal est le suivant : l'IMB400 est connecté à l'un des capteurs Imote2, celui-ci communique via un réseau sans fil avec le deuxième

1. Site internet : http ://www.tinyos.net/
2. Site internet : http ://nescc.sourceforge.net/
3. Fiche produit sur internet : http ://www.xbow.com/Products/productdetails.aspx ?sid=253
4. Fiche produit sur internet : http ://www.xbow.com/Products/productdetails.aspx ?sid=280
5. Fiche produit sur internet : http ://www.xbow.com/Products/productdetails.aspx ?sid=262

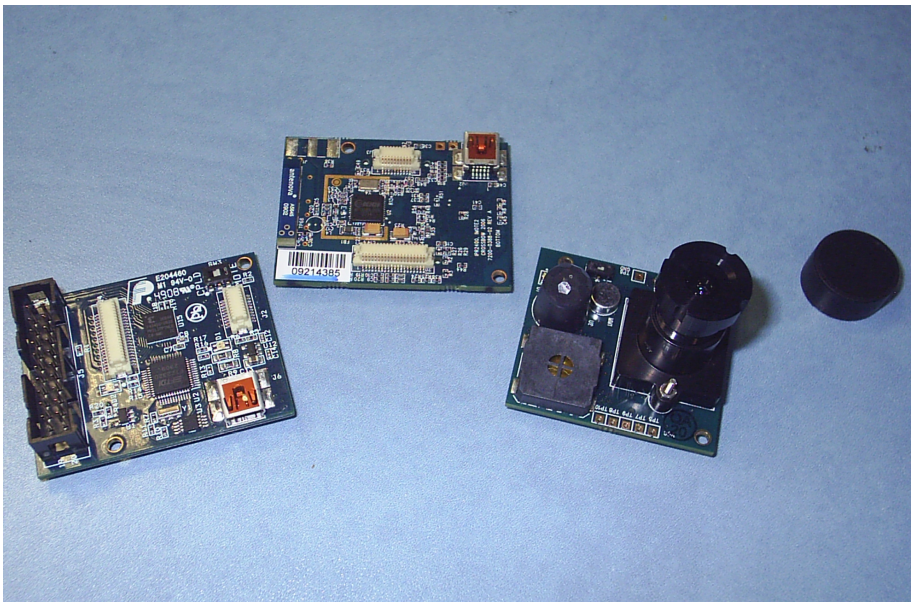

FIGURE 2.1 – De gauche à droite : IIB2400, IPR2400, IMB400

capteur qui est relié à l'ordinateur via un câble USB.

Malheureusement, il est impossible à l'heure actuelle de recevoir les données provenant de l'Imote2 sur un ordinateur tournant sur un système d'exploitation UNIX via la norme USB. Pour résoudre ce problème,il faut réaliser une émulation de port série (RS-232), c'est à dire "transformer" via logiciel le port USB en port série. L'utilisation de ce type de communication limite énormément les performances lors de la réception des données : le débit maximum en communication avec un ordinateur pour une connexion série est de 115.2 Kb/s (Pour comparaison, la vitesse minimale pour la norme USB 1.0 est de 1,5 Mb/s).

Au faible débit de la liaison s'ajoute également une autre limitation : d'après la documentation disponible sur internet sur le site du distributeur[2], la caméra dont dispose l'IMB400 permet en théorie de capturer jusqu'à 30 images par seconde, en couleur et à une résolution maximale de 640x480 (VGA). Sous TinyOS, ces spécifications ne sont malheureusement pas réalistes, la caméra n'étant que très partiellement supportée : premièrement, la couleur ne marche pas, ce qui nous limite aux images en niveaux de gris. Deuxièmement, cette carte multimédia gère la compression des images au format JPEG, ce qui permet d'obtenir des images de petite taille. Malheureusement, le support de la compression JPEG est tout simplement inexistant sous TinyOS[4]. On comprend vite qu'il faudra plusieurs secondes pour recevoir une seule image en niveaux de gris et en basse résolution (QVGA, soit 320x240), à cause du faible débit de transmission et de

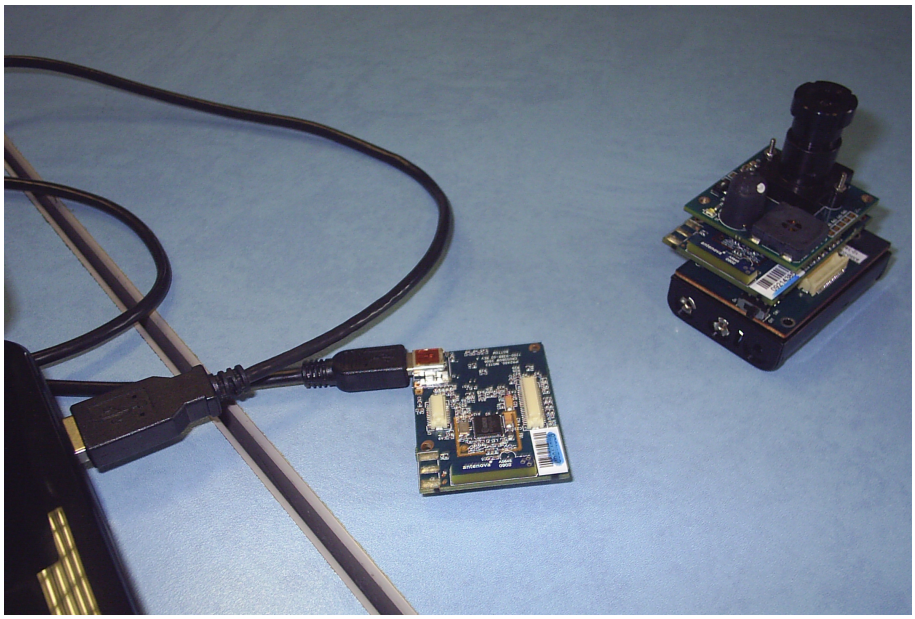

FIGURE 2.2 – Connexion idéale : un capteur qui reçoit les données provenant de l'autre capteur sur lequel est monté la caméra est directement connecté à l'ordinateur.

la taille des images non compressées : on est bien loin des 30 images par seconde théoriques.

Un autre problème fait son apparition lors du flashage [6] de l'Imote2 -opération nécessaire afin de permettre au capteur de commander la caméra et de réceptionner les données qu'elle envoie- : il n'est pas possible de flasher ce dernier via une connexion USB dans un environnement UNIX : cette méthode de flashage n'est tout simplement pas supporté à l'heure actuelle[3]. La seule solution possible est de se procurer un câble JTAG vers USB (dans notre cas, un Olimex ARM-USB-TINY [7]), et de le connecter à l'IIB2400 -la carte de débogage-, auquel l'on connecte le capteur.

2.2 État de l'art

Il n'y a que très peu de solutions ou de ressources sur le sujet sous TinyOS, en raison du manque de support de l'IMB400. Notamment, il n'y a aucun programme disponible permettant la visualisation en temps réel des images fournies

6. Modification du programme contenu dans la mémoire du capteur
7. http ://www.olimex.com/dev/arm-usb-tiny.html

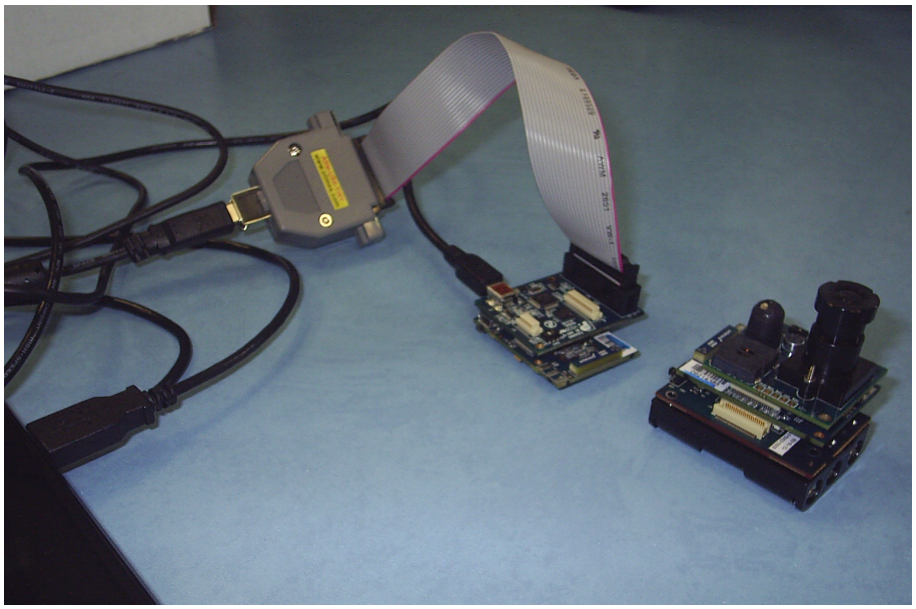

FIGURE 2.3 – Connexion nécessaire dans le cas de l'utilisation des environnements UNIX/TinyOS : la carte de débogage sur laquelle est greffé le capteur "base" est connectée à l'ordinateur en occupant deux ports USB.

par la caméra.

TinyOS fournit cependant un programme (écrit en Java, contribution d'utilisateurs) permettant la réception des données et l'enregistrement de celles-ci sous forme d'image au format PPM/PGM, un format non compressé de données graphiques. L'application ne permet cependant pas de visualiser directement l'image obtenue ni d'avoir une visualisation en continu et en temps réel du flux vidéo. L'interface du programme est très basique : une fenêtre qui se divise en deux parties : affichage de messages indiquant la progression de la réception d'une part, configuration de l'image souhaitée (compression JPEG, couleur, taille) et bouton de capture.

Notre travail s'est basé sur l'amélioration de ce programme, afin de permettre l'affichage dans la fenêtre de l'application des données reçues, et ce de façon continue, afin d'obtenir une "émulation" de temps-réel -il nous faut en effet tenir compte du faible débit de réception des données. Comme nous le verrons par la suite, ce choix a été motivé, en accord avec le responsable du projet, en raison du manque de temps, dut notamment au délai de réception des capteurs et du matériel nécessaire, mais également aux complications rencontrées lors des premières tentatives de flashage). Ceci remplace donc la visualisation telle qu'elle était prévue au départ, c'est dire sur navigateur web.

La priorité étant la visualisation des données reçues provenant de l'Imote2, cette dernière est donc devenue l'objectif principal devant la communication sans fil entre les capteurs.

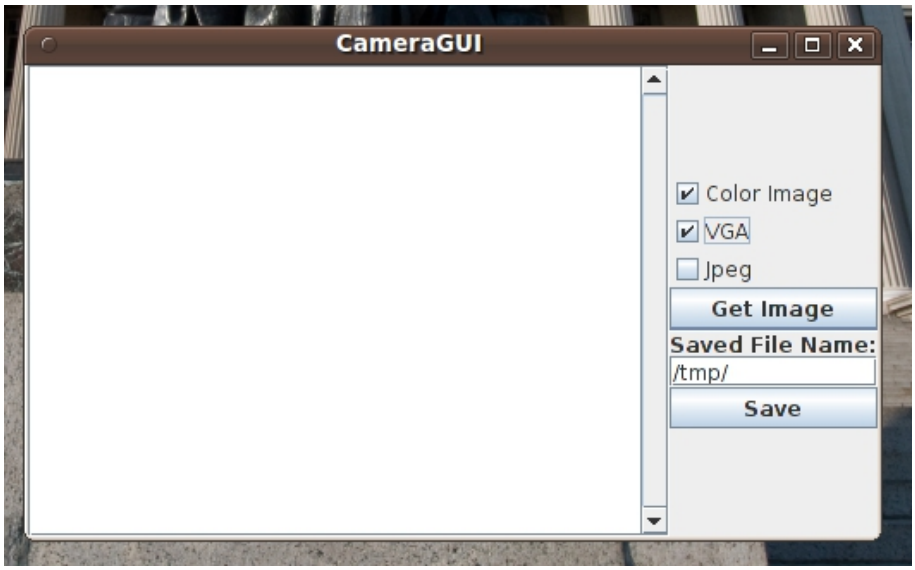

FIGURE 2.4 – L'application disponible au départ. A gauche, la fenêtre des messages, à droite les options et les commandes

Chapitre 3

Travail réalisé

3.1 Contexte

Comme entrevu précédemment, nous n'avons été en possession du matériel et du logiciel nécessaire à la réalisation du projet que presque deux mois après que nous ai été affecté le sujet (28 Janvier 2010). Cette attente s'explique par :

- Retard de livraison des capteurs (reçus le 8 Mars 2010 - retenus à la douane, leur disposition était prévue par notre responsable le 2 Mars 2010)

- La nécessité d'un câble JTAG pour le flashage des capteurs (et donc pour la réalisation des premiers tests) ne nous est apparu que plus tard (le 12 Mars 2010. Notre responsable de projet s'en était alors lui aussi déjà aperçu et venait de commander le câble.).

- Réception du câble JTAG le 15 Mars 2010.

- Malgré la possession de tout le matériel nécessaire, il nous est impossible de flasher les capteurs, en raison de problèmes logiciels. Après plusieurs tentatives (modification des fichiers de configuration, mise-à-jour des logiciels nécessaires et jusqu'à réinstallation du système), abandon de la version du système d'exploitation fournie par notre responsable pour une autre version qui celle-ci se révèlera fonctionnelle (25 Mars 2010).

Ce retard a eu inévitablement des répercussions sur les objectifs prévus initialement.

3.2 Site internet

Comme nous en avons pu faire l'expérience, l'installation n'est pas des plus aisées. Il faut avant tout avoir le matériel nécessaire, ce qui n'est pas forcement

rappelé explicitement dans les diverses documentations (C'est notamment vrai concernant le flashage du capteur sous UNIX : nous ne nous sommes rendus compte de la nécessité d'un câble JTAG tardivement.) De plus, ces documentations ne sont pas régulièrement mises-à-jour et sont depuis devenues obsolètes (liens non disponibles par exemple, utilisation de versions de programmes trop anciennes).

Il nous a été proposé par notre responsable de réaliser une page internet simple, afin de donner les étapes nécessaires au fonctionnement de l'ensemble capteur Imote2 et carte multimédia IMB400 sous TinyOS. Pour les raisons citées précédemment, cela nous a semblé une bonne idée. Ainsi, nous avons préparé un tutoriel dans le but de faciliter la configuration et la préparation du système pour ceux qui dans le futur auraient besoin de réaliser la même installation que nous. Le programme que nous avons réalisé sera également disponible, afin de permettre à des utilisateurs de le tester ou même de l'améliorer.

3.3 Récupération des données et affichage

Les données d'image transmises par le capteur utilisent le système GRBG, très utilisés dans la plupart des capteurs CMOS [1], qui font légion entre autres dans la téléphonie et la photographie. Ce système utilise deux valeurs de vert au lieu d'une, imitant la physiologie de l'il humain. Après conversion, on obtient des données utilisant le système RGB classique. Le capteur peut aussi n'envoyer que les niveaux de gris de chaque pixel si la couleur n'est pas sélectionnée.
Telles quelles, l'affichage et la sauvegarde de ces données dans un format d'image lisible est impossible. Il nous faut donc créer un format d'image à partir de ces données pour pouvoir en permettre l'affichage. Nous utilisons pour cela les formats PGM/PPM.

Un fichier PGM (portable graymap format) ou PPM (portable pixmap format) est en fait un fichier ASCII, contenant une en-tête définissant les propriétés d'une image (sa hauteur et sa largeur, le type - couleur ou niveau de gris dans les cas qui nous concernent-), suivies des valeurs définissant chacun des pixels.

Le choix de ces formats se justifie par la similarité de ces derniers avec les données reçues en provenance du capteur une fois converties dans le système RGB. En effet, celles-ci ne sont que des suites de valeurs codant chaque pixel. Il suffit donc de rajouter aux données obtenues l'en-tête correspondante - Taille et type de l'image- pour obtenir une image qui pourra alors être affichée.

La fenêtre du programme après modification est divisée en trois parties : affichage de l'image reçue, configuration de l'image souhaitée et boutons de

1. Complementary metal oxide semi-conductor

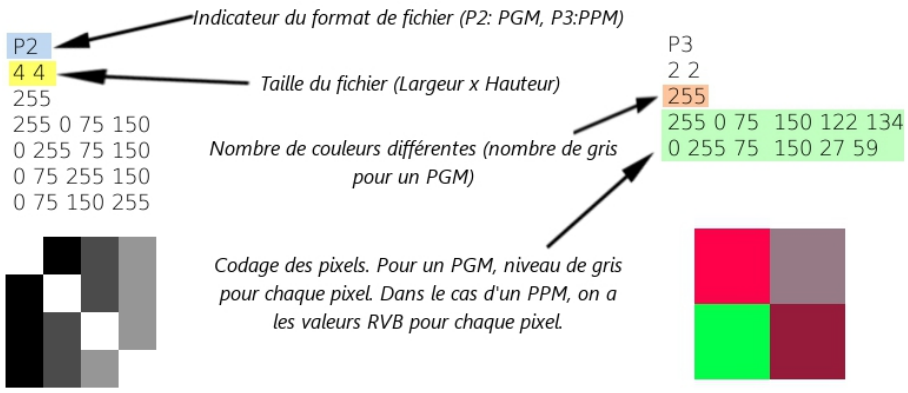

FIGURE 3.1 – Contenu des fichiers PPM et PGM

commande, zone de messages pour suivre la progression de la réception. Au niveau de la partie configuration, des choix ont été faits :

- Suppression de l'option de compression en JPEG, puisque non fonctionnelle (le programme ne permet de toute façons que d'afficher des images au format PPM/PGM)

- Liste des dimensions disponibles pour l'image (à l'origine, seule une case à cocher servait à choisir la taille 640x480, mais sans spécifier l'autre taille disponible. La modification a été faite pour plus de clarté)

- Liste des modes de couleurs disponibles (pour les mêmes raisons que précédemment, seule était disponible une case à cocher pour la couleur)

Il est à noter que nous avons décidé de ne pas supprimer le mode couleur. En effet, bien que celui-ci ne marche pas, il produit tout de même des images en niveaux de gris, bien que plus sombres qu'en sélectionnant le mode "niveaux de gris". De plus le programme est codé de sorte qu'il permette l'affichage des images de couleur, ainsi, si la couleur est un jour disponible, le programme saura afficher l'image correctement sans modifications nécessaires.

Le programme fonctionne sur deux modes : le premier permet de ne réaliser qu'une seule prise, avec possibilité par la suite d'enregistrer l'image capturée. Le deuxième mode récupère en continu des images. Pour les raisons expliquées précédemment, il se passe plusieurs secondes avant obtention d'une image.

Des choix ont été faits également dans la manière dont sont affichées les images. Plusieurs choix s'offraient à nous :

- Affichage de l'image lorsque la réception est complète.

– Affichage des donnes reçues "en direct", recouvrement de l'image précédente par les données de l'image suivante.

– Affichage de l'image courante dans une fenêtre, de l'image précédente dans une autre fenêtre.

La première solution a été rapidement éliminée, le résultat aurait été trop "statique" et n'aurait pas donné l'impression d'un fonctionnement en temps-réel. La troisième solution l'a emportée sur la deuxième : en effet, l'image précédente est visible plus longtemps dans le cas d'un affichage sur une deuxième fenêtre et, étant donné le temps de réception d'une seule image, il semblait plus judicieux de garder cette image le plus longtemps possible pour comparaison avec l'image suivante.

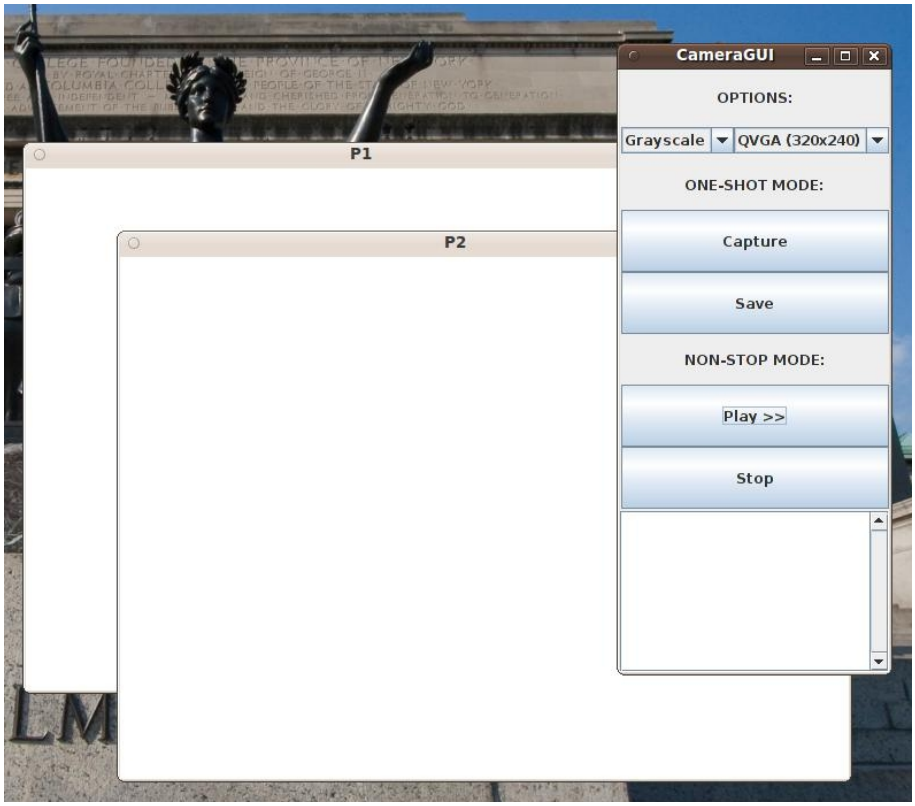

FIGURE 3.2 – Le programme une fois amélioré.

Voici donc résumées les améliorations que nous avons apporté au programme de départ :

- Au niveau des fonctionnalités :
  - Possibilité, en plus de ne faire qu'une prise simple, de recevoir en continu les images provenant de la caméra (temps-réel : une nouvelle image n'est capturée que lorsque la précédente a été entièrement reçue -pas d'"accumulation" d'images-), avec arrêt possible.
  - Affichage de l'image en cours de réception dans une fenêtre séparée. Dans le cas où l'on demande une capture continue, une image une fois entièrement reçue est déplacée dans une deuxième fenêtre, tandis que l'on voit apparaitre progressivement l'image suivante dans la première fenêtre.
- Au niveau de l'interface utilisateur :
  - Suppression de l'option de compression JPEG, non fonctionnelle.
  - Affichage explicite des modes disponibles (Niveaux de gris, Couleur) et des dimensions disponibles (QVGA, VGA).
  - Boîte de dialogue lors de la sauvegarde de l'image en mode capture simple.

FIGURE 3.3 – Exemple d'une capture provenant de l'Imote2.

3.4 Communication sans fil

En raison du manque de temps disponible, la communication sans fil entre capteurs n'est pas fonctionnelle au moment d'écrire ce rapport. Nous prévoyons cependant, dans la limite du possible, d'essayer de l'implémenter, non seulement parce qu'elle rentrait dans les objectifs initialement prévus, mais également parce que la communication sans fil est un des enjeux principaux de ce type de capteurs. Nous avons également bonne espérance que l'interface Java que nous avons développée soit toujours compatible dans le cas d'une utilisation impliquant le sans-fil, et ce sans modifications : les commandes aux capteurs vidéos n'ont pas a être changées, le capteur servant de base n'étant qu'un pont vers le capteur vidéo (ce qui reviendrait au même que le cas actuel).

Chapitre 4

Conclusion

Certains des objectifs définit initialement n'ont pu être atteints (communication sans fil entre capteurs, visualisation sur un navigateur internet). Cependant, les objectifs prioritaires redéfinis par la suite ont été atteints (Principalement, visualisation en continu d'images en provenance du capteur -dans la limite des limitations indues au faible débit de réception- sur une version améliorée du programme fournit par TinyOS). La non-existence de connexion sans fil entre les capteurs à l'écriture de ce rapport est cependant fort dommage, puisque il s'agit d'une part importante des avantages d'utilisation de ces capteurs... Nous espérons tout de même pouvoir l'implémenter prochainement.

Nous sommes tout de même satisfaits de l'amélioration que nous avons réalisés au programme de départ. Il n'y a pour l'instant et à notre connaissance aucune autre solution disponible permettant une visualisation en temps réel d'un capteur vidéo Imote2 sous TinyOS.

L'utilisation de l'environnement TinyOS pour la visualisation d'un flux vidéo provenant d'un capteur vidéo Imote2 n'est cependant pas une solution viable, tout du moins tant que l'on sera obligé de faire transiter les données par émulation d'une connexion série, bien trop lente. Cependant, si un jour cette limitation venait à être levée, l'utilisation de l'environnement TinyOS deviendrait beaucoup plus recommandable, déjà parce qu'il est adapté à ce type de capteurs sans fil et par le fait qu'il s'agit d'un projet open-source.

Bibliographie

[1] Pham, C, *http ://web.univ-pau.fr/ cpham/PROJETS/WSN/Home.html*

[2] Crossbow Technology, Inc., *Imote2 Multimedia IMB400*, *http ://www.xbow.com/Products/Product_pdf_files_Wireless_pdf/Imote2_IMB400 _Preliminary.pdf*

[3] TinyOS contributors, *Imote2*, *http ://docs.tinyos.net/index.php/Imote2*

[4] Crossbow Technology, Inc., *Where can I find the drivers for the IMB400 Imote2 camera board ?*, http ://www.xbow.com/Support/wobjectDetail.aspx ?id=50160000000LW4hAAGtype= Solutionpage=0

Annexe A

Code NesC (extraits)

Voici quelques extraits du code NesC qui tourne sur le capteur Imote2 sur lequel est attachée la camera.

Réception des paramètres de l'image (envoyés depuis le programme JAVA : taille de l'image, couleur, compression JPEG...) :

```
event message_t *CmdReceive.receive(message_t *msg, void *payload, uint8_t len) {
cmd_msg_t *cmdMsg = (cmd_msg_t *)payload;
if ((cmdMsg->cmd & 0x01) == 0x01) { // color
img_stat.type |= IMG_COL;
}
else {
     img_stat.type &= ~IMG_COL;
}
if ((cmdMsg->cmd & 0x02) == 0x02) { // size
img_size = SIZE_VGA;
     img_stat.width = VGA_WIDTH;
     img_stat.height = VGA_HEIGHT;
}
else {
     img_size = SIZE_QVGA;
    img_stat.width = QVGA_WIDTH;
     img_stat.height = QVGA_HEIGHT;
}
if ((cmdMsg->cmd & 0x04) == 0x04) { // compression
     img_stat.type |= IMG_JPG;
}
else {
     img_stat.type &= ~IMG_JPG;
}
post acquireTask();
```

```
return msg;
}
```

Récupération des données de la caméra : dans le cas d'une image en couleur, demande du format RGB565 (5 bits pour le Rouge, 6 bits pour le vert, 5 bites pour le bleu) :

```
task void acquireTask() {
img_stat.timeAcq = -call Timer0.getNow();
if (!(img_stat.type & IMG_COL)) {
frame = call XbowCam.acquire(COLOR_UYVY, (void*)BASE_FRAME_ADDRESS, img_size, 0); //sub
}
else {
     frame = call XbowCam.acquire(COLOR_RGB565, (void*)BASE_FRAME_ADDRESS, img_size, 0)
}
}
```

Formatage des données à envoyer : les données d'image sont envoyées dans un sytème GRBG, cela permet de n'envoyer que 2 bytes (16 bits, 5 pour coder le rouge, 5 pour coder le bleu et 6 pour coder le vert) au lieu de 3 dans le cas d'un RGB classique :

```
task void fixAcqBuffer() {
int32_t i;
    if (img_stat.type & IMG_JPG) {
     // COL non-JPG images don't need fix, it's more efficient to xmit 2 bytes than 3
for (i = (frame->header->size / 2 - 1); i >= 0; i--) {
   uint8_t b1 = frame->buf[2 * i];
   uint8_t b2 = frame->buf[2 * i + 1];
   frame->buf[3 * i] = b2 & 0xF8; // red
   frame->buf[3 * i + 1] = ((b2 & 0x07) << 5) | ((b1 & 0xE0) >> 3); // green
   frame->buf[3 * i + 2] = (b1 & 0x1F) << 3; // blue
}
frame->header->size = frame->header->size / 2 * 3;
    }
}
```

Annexe B

Code JAVA (extraits)

B.1 Récupération d'image

Cet extrait de code issu du programme Java de départ permet de générer un format d'image valide à partir des données reçues du capteur Imote2.

Tout d'abord, création des en-têtes nécessaires. Ceux-ci contiennent, comme vu précédemment, un identifiant du format (P2 pour une image en niveaux de gris, P3 pour une image en couleur), les dimensions, et le nombre de couleurs ou de niveaux de gris différents (ici fixé à 255) :

```
DataOutputStream stream = null;
OutputStreamWriter writer = null;
String pgmHeader =  "P2\r\n" + max_x + " " + max_y + "\r\n" + "255\r\n";
String ppmHeader =  "P3\r\n" + max_x + " " + max_y + "\r\n" + "255\r\n";
```

Par la suite, ecrite de l'en-tête correspondant au type de l'image. On remarque que la partie de code concernant la compression JPEG est toujours présente, bien que non fonctionnelle (la compression JPEG est déselectionnée par défaut) :

```
if (isCompression.isSelected()) {
stream = new DataOutputStream(fostr);
}
else {
writer = new OutputStreamWriter(fostr);
if (colorList.getSelectedItem().toString().equals(grayscale))
writer.write(pgmHeader);
else
    writer.write(ppmHeader);
}
```

Les données RAW provenant de l'Imote2 utilisent un système GRBG (filtre Bayer) : pour un pixel, on a quatre photocites : un pour la couleur bleue, un autre pour le rouge, et deux pour le vert. Cet arrangement tient compte de la sensibilité de la vision humaine. Il faut donc reconstituer les informations de couleur.

Initialisation des variables (red_byte : $lavaleurquicodeleniveauderouge$, $blue_byte$: $lavaleurquicodeleniveaudebleu$, $green_byte$: $lavaleurquicodeleniveaudevert$, $green_loetgreen_hi$: $lesdeuxverts$).

```
int cnt_row = 1;
int red_byte = 0;
int blue_byte = 0;
int green_byte = 0;
int green_lo = 0;
int green_hi = 0;
boolean is_first_byte = true;
for (int i = 0; i < data.size(); i++)
{
for (int j = 0; j < 64; j++)
{
int curr_byte = 0;
if (j < data.get(i).length)
curr_byte = (data.get(i)[j] & 0xFF);
else if (i == data.size() - 1)
break;
```

Si l'image est en niveau de gris, on garde les données sans modifications.

```
if (isCompression.isSelected())
{
stream.writeByte(curr_byte);
}
else if (colorList.getSelectedItem().toString().equals(grayscale))
{
writer.write((curr_byte & 0xFF) + " ");
if (++cnt_row >= max_x)
{
cnt_row = 0;
writer.write("\r\n");
    }
}
```

S'il s'agit d'une image en couleur, les 8 premiers bits codent les valeurs GR et les 8 bits suivants les valeurs BG. On récupère donc, par masquage des bits, les valeurs correspondantes aux couleurs souhaitées : les 5 premiers bites du premier byte correspondent au rouge, les 3 derniers correspondent au premier

vert. Les 3 premiers bits du deuxième byte correspondent au deuxième vert, les 5 derniers correspondent au bleu. On réalise par la suite des déplacements de bits afin d'obtenir les couleurs correspondantes dans le système RGB (pour obtenir le vert, on fait un OU entre $\text{green}_h ietgreen_l o$)

```
else
{
if (is_first_byte)
{
is_first_byte = false;
red_byte = (curr_byte & 0x1F);
green_lo = (curr_byte & 0xE0);
green_lo = ((green_lo >> 5) & 0x07);
}
else
{
blue_byte = ((curr_byte >> 3) & 0x1F);
green_hi = (curr_byte & 0x07);
green_hi = (green_hi << 3);
green_byte = (green_lo | green_hi);
red_byte <<= 3;
blue_byte <<= 3;
green_byte <<= 2;
writer.write((red_byte & 0xFF) + " " + (green_byte&0xFF) + " " + (blue_byte&0xFF) + " ");
if (++cnt_row >= max_x)
{
    cnt_row = 0;
    writer.write("\r\n");
}
is_first_byte = true;
}
}
}
}
```

B.2 Affichage d'une image

Voici le code que nous utilisons pour afficher l'image dans l'interface.

La première partie est semblable au code vu précédemment :

```
public void updateGraphic(short[] tmp, int imgType)
{
int red_byte = 0;
    int blue_byte = 0;
int green_byte = 0;
```

```
int green_lo = 0;
int green_hi = 0;
boolean first_byte=true;
Points = new Vector();
for (int xx=0; xx<tmp.length; xx++)
{
if (curr_x == max_x)
{
curr_x = 0;
    curr_y ++;
}
int curr_byte=0;
curr_byte = (tmp[xx] & 0xFF);
```

S'il s'agit d'une image grise, on recopie trois fois en tant que valeurs RGB le byte courant (lorsque R=G=B, on obtient un niveau de gris) :

```
if (imgType == 2)
{
int b = (curr_byte & 0xFF);
    Color cl = new Color(b,b,b);
            Points.add(new PointTrace ( curr_x , curr_y , cl ));
            curr_x++;
}
```

Sinon enregistrement des valeurs R, G et B :

```
else
{
if (first_byte)
    {
first_byte = false;
   red_byte=(curr_byte & 0x1F);
green_lo=(curr_byte & 0xE0);
green_lo=((green_lo >> 5) & 0x07);
}
else
{
    blue_byte = ((curr_byte >> 3) & 0X1F);
    green_hi = (curr_byte & 0x07);
    green_hi = (green_hi << 3);
    green_byte = (green_lo | green_hi);
    red_byte <<= 3;
    blue_byte <<= 3;
    green_byte <<= 2;
    int r,v,b;
    r = (red_byte & 0xFF);
```

```
    v = (green_byte & 0xFF);
    b = (blue_byte & 0xFF);
    Color cl = new Color(r,v,b);
    Points.add(new PointTrace ( curr_x , curr_y , cl ));
    first_byte = true;
    curr_x++;
}
}
}
draw.updateGraphics(Points);
}
```