Projet de développement
Gopiwan

Rapport Technique
Master 1 *Web Multimédia Réseau*

David BARRAT
Nicolas BLOYET
Nailya BOGROVA
Guillaume CLAUDIC

14 mai 2015

Table des matières

1 Mise en marche

1.1 Démarrer le GoPiGo

La mise en route du robot contrôlé à distance se passe sans que l'utilisateur n'ait à se préoccuper d'aucun paramétrage. Lors de la livraison du robot, il y aura tout le nécessaire pour que le robot fonctionne "out-of-the-box". En effet, grâce à des scripts de démarrage préalablement installés, le GoPiGo démarrera dès qu'il sera sous tension :

— le wifi en mode hotspot

— le serveur http Jetty

Le Pi sera donc dès lors accessible à l'adresse *192.168.42.1* sur le réseau wi-fi de SSID *GOPIGO* (réseau ouvert).

1.2 Connecter un appareil sur le GoPiGo

Comme précisé précédemment, le GoPiGo déploie à son démarrage un réseau wi-fi ad-hoc, au sein duquel il dispose de l'adresse *192.168.42.1.* Pour y connecter un appareil, il suffit de joindre le réseau "GOPIGO", de préférence avec une ip statique appartenant au réseau *192.168.42.0 / 255.255.255.0*, le serveur dhcp du GoPiGo étant malheureusement hasardeux...

A partir de là, le GoPiGo est ouvert à tout contact par requête HTTP sur le port *8080*, par exemple l'accès à l'interface embarquée, qui se fera dans un navigateur à l'adresse :

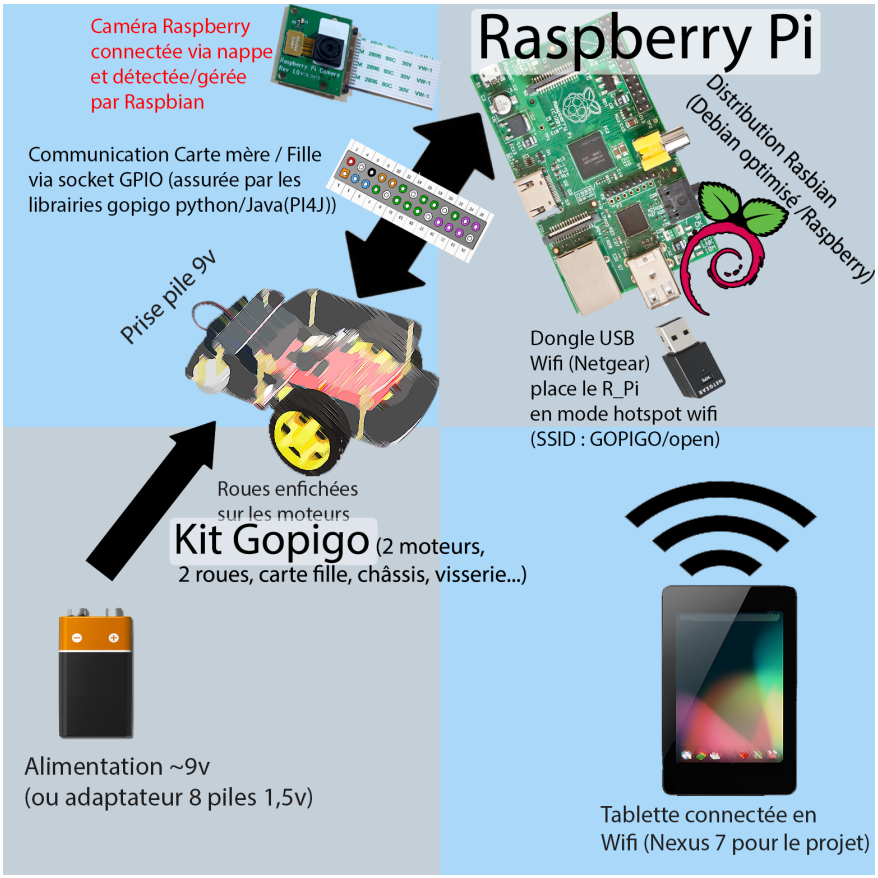
http ://192.168.42.1 :8080/interface

1.3 Note concernant le Wi-Fi

En l'état actuel des choses, le wi-fi est relativement instable, c'est à dire que la connexion peut se perdre, et que le serveur DHCP est plutot hasardeux. Peut-être est ce en raison du matériel, peut être est ce dû à de mauvais réglages de notre part...

Pour le savoir il faudrait essayer avec un autre dongle.

2 Intéractions matérielles



3 Dépendances à Python

Lors du développement de notre application, nous avons été confrontés à des problèmes techniques : matériels, mais aussi logiciels ; En effet, au début du projet, les librairies Java n'existaient pas encore, on a donc opté pour l'utilisation des librairies python Gopigo déjà existantes, qu'on appelle de deux façons différentes :

— Un démon python pour des communications par socket TCP

— Des appels directs à l'interpréteur python au cours du programme

Dans le premier cas, on lance un démon python à l'écoute sur un port TCP qui se charge de recevoir des appels qu'on lui envoie et se charge de lancer les commandes correspondantes sur le pi. C'est une méthode qui nécessite donc l'exécution continue d'un programme supplémentaire. L'autre solution permet de s'affranchir de démon, en faisant appel à un interpréteur python à chaque appel, exécutant alors des scripts préalablement placés dans */home/pi/scripts*.

Chaque méthode à ses avantages et ses inconvénients, aussi nous avons choisi de mettre les deux à disposition, à choisir selon l'utilisation. En effet, dans le cas ou le robot ne bouge pas souvent, il est plus économe de faire des appels ponctuels à l'interpréteur sans s'encombrer d'un processus supplémentaire, tandis que si les mouvements sont fréquents, le démon se révèlera plus pertinent.

La question des coûts était primordiale dans ce projet, car le GoPiGo dispose d'une part de peu de ressources, le CPU étant de faible puissance, et devant potentiellement gérer un streaming vidéo, et d'autre part, il s'agit d'un système fonctionnant sur batterie, l'autonomie de l'appareil est donc en jeu.

Il est donc possible de changer de mode de communication en cours même d'utilisation.

4 Interface Client

4.1 Interface embarquée

Le GoPiGo dispose d'une interface embarquée, qui se trouve être une page html simple, permettant très basiquement d'envoyer les requêtes adéquates au contrôle du GoPiGo via des boutons, et d'afficher son flux vidéo. Le choix d'html s'est fait de part pour son universalité, mais aussi de sa simplicité.

En effet, du fait de l'API que nous avons developpé et dont nous reparlerons ultérieurement, la seule chose que l'interface ait à faire est d'envoyer des requêtes HTTP à l'adresse et au port adéquat. Cette charge est donc très limitée et peut se réaliser très rapidement, notamment au sein d'une page HTML, avec le mécanisme AJAX.

Or, une page html équipée de javascript offre une compatibilité très large de matériel : tout appareil disposant d'un navigateur est ainsi en mesure d'accéder à cette interface et de pouvoir contrôler le robot, que ce soit un téléphone de n'importe quelle marque, un ordinateur traditionnel, ou autre...

Il n'y a donc de fait aucune contrainte de plate-forme.

Cette interface embarquée est accessible à l'adresse :

http ://192.168.42.1 :8080/interface

4.2 Mode d'emploi de l'interface embarquée

4.2.1 Barre d'adresse

Comme énoncé précedemment, pour accéder à l'interface Gopigo, il est nécessaire de saisir l'adresse de l'interface wlan0 (par défaut) du raspberry pi. Ici, l'adresse du pi est 192.168.42.1 et le serveur tourne sur le port 8080. Sur la capture d'écran, le path /interface n'apparait pas, mais il faut néanmoins le saisir après le port pour accéder à la page.

4.2.2 Vue caméra

Lorsque l'utilisateur charge l'interface (et que tout s'est bien passé sur le pi), il est censé avoir l'affichage perçu par la caméra du robot.

4.2.3 Commandes robot/ camera

La troisième partie de l'interface concerne les commandes utilisateur,

4.3 Directions

— Gauche, permet de faire tourner la roue gauche du robot, qui aura pour incidence de le faire tourner vers la gauche.
— Droite, permet de faire tourner la roue droite du robot, qui aura pour incidence de le faire tourner vers la droite.
— Avancer, permet de faire tourner les deux roues du robot vers l'avant, qui aura pour incidence de le faire avancer.
— Reculer, permet de faire tourner les deux roues du robot vers l'arrière, qui aura pour incidence de le faire reculer.
— Stop, permet d'arrêter la rotation de la ou des roues, si elles sont en activité.
— Random, fonctionnalité permettant d'executer une des commandes ci-dessus, mais laquelle ?

4.4 Camera

— CameraOn, permet d'activer la caméra du robot.
— CameraOff, permet de désactiver la caméra du robot.

4.5 Demi-tours

— Rotation horaire, permet de faire faire au robot un demi-tour en sens dextrogyre (sens des aiguilles d'une montre).
— Rotation anti-horaire, permet de faire faire au robot un demi-tour en sens lévogyre (sens inverse des aiguilles d'une montre).

4.6 Paramètres caméra

— Résolution, permet de modifier la résolution des images envoyées par le robot (entre 240p à Full HD, 360p conseillée).
— Frames par seconde, permet de modifier le nombre d'image(s) envoyée(s) par seconde, par le robot, et étant données les contraintes matérielles, les valeurs envisageables oscillent entre 1FPS et 10FPS.

4.7 Vitesse du robot

Ici, nous avons opté pour un slider afin de choisir la vitesse du robot, plutôt que de laisser à l'utilisateur la possibilité de saisir des valeurs de vitesse incohérentes. On peut choisir entre une vitesse allant de 0 à 50 (bien entendu, ce ne sont pas des km/h, c'est une unité arbitraire propre à GoPiGo).

4.8 Développement d'une interface grâce à l'API

En dehors de cette interface embarquée très simple, il est possible de réaliser très simplement d'autres interfaces grâce à l'API REST mise a disposition. En effet, à partir du moment où un programme peut envoyer une requête HTTP, il

peut se servir de cette API, et n'importe qui ayant lu la documentation fournie peut donc développer sa propre interface, sous la forme voulue (Page HTML, application mobile, logiciel desktop "classique", etc...).

La documentation relative à cette API est bien entendu fournie.

5 Programme serveur

Le programme serveur fourni est en fait un intergiciel, faisant l'interface entre des appels provenant d'interfaces clientes, et la librairie GoPiGo Python.

De façon très schématique, il met à disposition les ressources proposées par le GoPiGo selon le style d'architecture REST, et traduit donc les requêtes HTTP des clients en appels à l'API GoPiGo Python.

5.1 Fonctionnement

Nous avons utilisé deux librairies externes pour le developpement du serveur :

— *Jetty*, projet de la fondation Eclipse

— *Jersey*, implémentation de la spécification *JAX-RS* par Oracle

Jetty est un serveur HTTP et moteur de servlet, qui nous sert donc à créer un serveur HTTP à l'écoute sur le port *8080*, et dont certaines classes constitueront des servlet, qui en l'occurrence délivreront le résultat de l'éxécution des commandes.

Ces servlet seront annotés grâce à *Jersey*, qui nous permettront de définir les verbes HTTP adéquats, les *paths* relatifs à chaque fonction, mais aussi le type de données produites.

Pour information, les fonctions de mouvement délivrent une réponse en texte brut, tandis que l'interface est délivrée en html, et les frames vidéo en bitmap.

Les annotations Jersey permettent donc d'obtenir un code combinant lisibilité et élégance et adaptable aux souhaits du développeur.

Ci-dessous un exemple commenté d'utilisation de telles annotations :

```
@Path("/gopigo")          // on definit le path d'acces au servlet
public class MoveService {
 // ...
    @GET       // cette fonction répondra au verbe GET...
    @Path("/stop")    // ... utilisé a l'adresse /gopigo/stop...
    @Produces(MediaType.TEXT_PLAIN) // ... et rendra sa réponse sous
                                    // forme de texte brut
    public String stop() throws IOException {

        Globals.gopigo.stop();       // appel au matériel
        return "Stopped !\n" ;       // réponse rendue
    }
```

5.2 Architecture

Le programme a été fait de façon la plus modulaire possible. Les classes ont été regroupées par package suivant leur rôle.

5.2.1 Packages

Le programme serveur est fourni sous forme d'un jar éxécutable, organisé comme suit :

```
src/
├── server/ ..... Sources serveur
│   ├── Globals.java
│   ├── HttpServer.java
│   ├── Launcher.java
│   ├── daemons/ ..... Processus d'arrière plan
│   │   ├── DaemonPython.java
│   │   └── DaemonStreaming.java
│   ├── gopigo/ ..... Interface API / matériel
│   │   ├── GoPiGo.java
│   │   ├── GoPiGoMoves.java
│   │   └── GoPiGoStreaming.java
│   ├── python/ ..... Scripts python
│   │   └── macros.py
│   └── rest/ ..... Services web (API)
│       ├── HTMLService.java
│       ├── MoveService.java
│       ├── SettingsService.java
│       └── StreamService.java
└── client/ ..... Sources client
    └── index.html
```

Chaque package a donc une fonctionnalité bien précise, de façon à rendre chaque partie la plus indépendante possible.

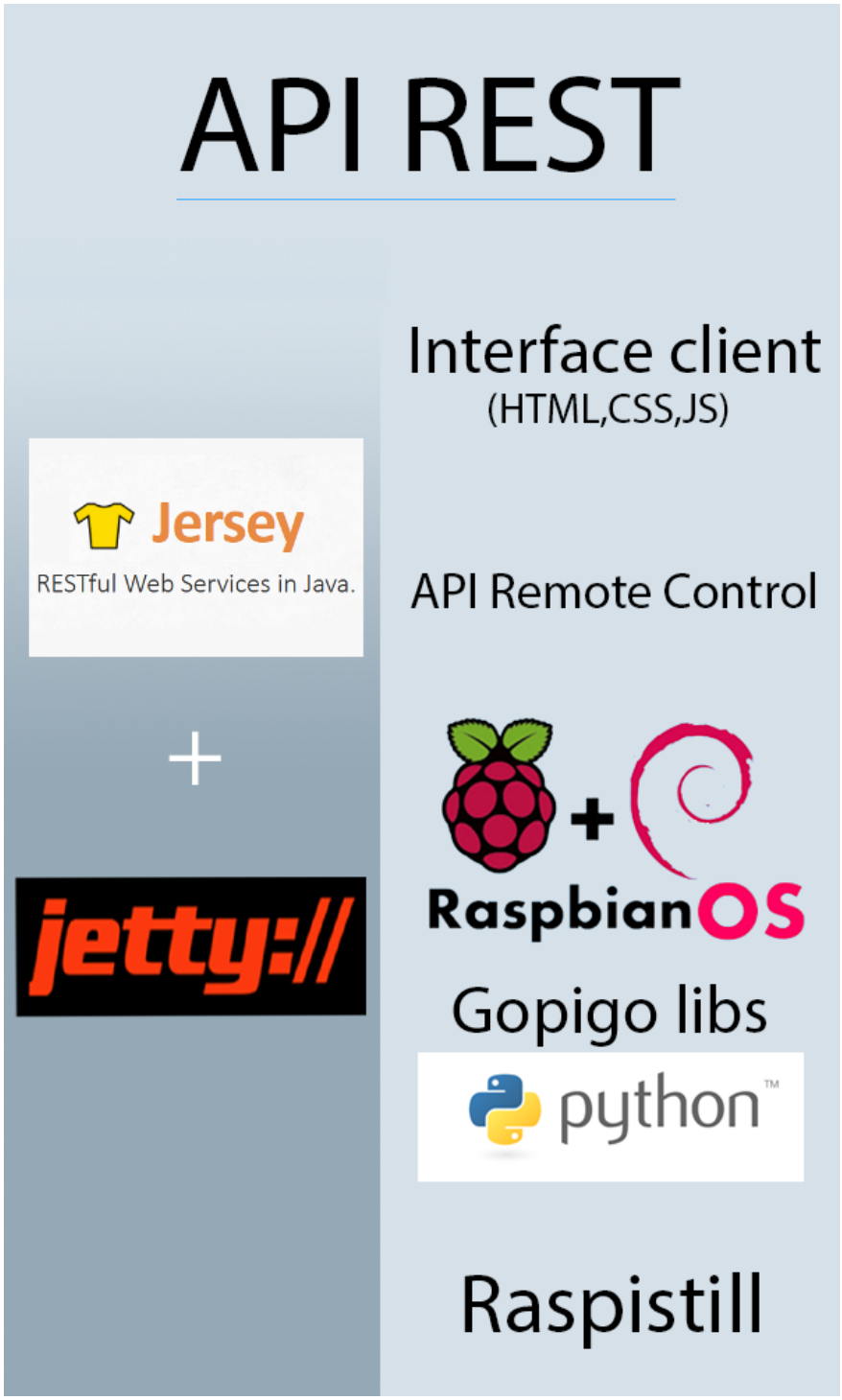
Le package *dameons* contient tous les processus d'arrière-plan nécéssaires au bon fonctionnement du GoPiGo. Ces classes sont typiquement instanciées par *server.gopigo.GoPiGo*. Il se pourrait qu'à terme, le contenu de ce package soit revu à la baisse.

Cette classe *GoPiGo*, qui joue le rôle d'interface entre les appels de l'API et le matériel, est en quelque sorte le point central du programme, le point d'échange des autres packages. L'utilisation de cette classe se justifie du fait que le mode de communication avec le matériel peut évoluer au cours du temps, suivant les librairies utilisées pour communiquer à la carte GoPiGo.

Le package python contient tous les scripts nécésssaires au fonctionnement du GoPiGo, et son contenu doit être copié au préalable dans */home/pi/scripts*.

Enfin, le package *rest* contient toutes les classes faisant office de servlet, et contitue donc le point d'entrée des requêtes client. Il faut noter que ces *Web Services* sont eux aussi indépendants de la librairie utilisée pour communiquer avec le matériel, et que de ce fait, ils ne font que faire suivre les fonctions de l'API à la fonction correspondante de *server.gopigo.GoPiGo*.

5.2.2 Schéma d'architecture



6 Evolution possible

Comme il l'a été énoncé dans ce rapport, le code de programme est le plus modulaire possible, et est de ce fait facilement adaptable à librairie de communication GoPiGo pour Java, disponible depuis peu.

Il suffira en fait en très grande partie à remplacer/rajouter les appels en python étant dans la classe *server.gopigo.GoPiGo* par des appels Java correspondants.

Cet intergiciel est donc facilement adaptable, sans pour autant devoir changer la partie des web services, ce qui serait dommageable si des interfaces basées sur notre API voient le jour.