TP réseau : contrôle à distance d'un moteur pas à pas

Pierre-Louis Guhur

May 4, 2017

Abstract

1 Introduction

Le DarwiPod est difficile à contrôler physiquement. Dans un premier temps, on peut se satisfaire de le contrôler à distance en lui ajoutant une puce WiFi : la carte ESP8266, très appréciée en domotique pour son prix et sa faible consommation énergétique. Le robot Darwin-Op pourra alors transmettre ses ordres par sa propre carte WiFi. En même temps, le robot n'est malheureusement pas encore autonome, alors un opérateur doit lui donner des ordres pour le déplacer.

L'objectif de ce TP est de réaliser ces interfaces entre l'opérateur et le robot (à travers le protocole HTTP), puis entre le robot et le gyropode (à travers le protocole TCP). Plutôt que de travailler directement sur eux (ce qui est difficile quand plusieurs groupes de TP travaillent en même temps), on propose de simuler la carte WiFi du robot par celle de son ordinateur, et le gyropode par un simple moteur pas-à-pas.

L'une des difficultés est de conserver la connexion ouverte entre le robot et le gyropode, pour réduire les temps de réaction du robot.

Un premier binôme a en charge la gestion logicielle des réseaux, tandis que le second binôme gère la gestion matérielle des réseaux. Les binômes devront néanmoins s'entraider pour avancer dans leur développement et obtenir le livrable final.

2 Ressources

2.0.1 Ressources

Ruby	https://www.ruby-lang.org/en/documentation/quickstart/
Socket	https://ruby-doc.org/stdlib-1.9.3/libdoc/socket/rdoc/Socket.html
Git	https://try.github.io/levels/1/challenges/1
TP	https://github.com/plguhur/eea-ens-paris-saclay/tree/master/Informatique/TP-module-wifi
Documentation	ESP8266ModuleV1.pdf, ESP8266ATCommandsSet.pdf, ESP8266_WiFi_Module_Quick_Start_Guide_v_1.0.4.pdf
nweb	https://github.com/ankushagarwal/nweb
Matériels	Poste de développement sous Unix avec une carte WiFi, ESP8266, adapteur de niveau, carte Leonardo
Logiciels	Arduino IDE, éditeur texte, terminal

3 Pré-requis

3.1 Langage Ruby

Le Ruby est un langage de programmation dynamique, *open source* réflectif, et orienté objet. Il a été développé dans les années 90 par Yukihiro "Matz" Matsumoto, qui espère que "Ruby aide chaque développeur à être productif, à apprécier la programmation et à s'en réjouir" [**?**].

Quelques éléments de syntaxe sont indiqués ici, tandis que le lecteur avisé regardera la documentation [**?**] pour avoir plus d'informations :

- Une fonction est définie par :

```
bluedef functionName(arg1, ..., argN)
    ....
blueend
```

  Son nom peut contenir un point d'interrogation ou un point d'exclamation. Pour appeler la fonction, il n'y a pas besoin de préciser les parenthèses.

- Une classe est définie par :

```
blueclass className
    attr_accessor :varName magenta#magenta magentavariablemagenta r

    bluedef initialize(arg,... ) magenta#magenta magentaconstructe
        ...
    blueend
    ....
blueend
```

3.2 Les sockets

Les sockets sont la base des éléments de communication entre deux processus. Blablabla...

3.3 Protocoles Internet

Le modèle OSI permet de découper les réseaux selon 7 différentes couches pour représenter le niveau d'abstraction. Les couches sont :

1. physique : comment les bits sont tranportés ? WiFi, câble éthernet, ...
2. liaison : commet les trames trouvent leur destinataire ? Adresse MAC, ...
3. réseau : comment les paquets trouvent leur chemin parmis les routeurs ? Protocole IPv4, IPv6, etc.
4. transport : comment les segments sont transportés sur le réseaau ? TCP (avec une connexion), UDP (pas de connexion)
5. présentation : sous quel format les fichiers comme les images sont présentés ?
6. session : comment garantir que le client soit le bon ?
7. application : comment les donnés sont utilisées ? Protocoles HTTP, FTP, SMTP, ...

Le module ESP8266 permet de créer ou de se connecter à un protocole TCP. On pourrait utiliser une surcouche comme Telnet, mais afin de diminuer les temps de communication, on préfère utiliser directement le protocole TCP. En outre, la communication entre le ESP8266 et le serveur doit être laissée ouverte pour ne pas perdre de temps à l'initialiser à chaque fois.

4 Premier binôme

Cette partie réalise le traitement logiciel des réseaux.

4.1 Objectifs

On va, dans un premier temps, programmer la partie serveur qui permet de récupérer des requêtes d'un serveur web (on prendra nweb) et de les transmettre à une socket simulant l'ESP8266.

4.2 Procédure

1. Utilisez la version modifiée de nweb pour accepter des requêtes CGI et exécuter un script : https://github.com/plguhur/eea-ens-paris-saclay/tree/master/Informatique/TP-module-wifi/nweb

2. Chargez l'interface web et observez quel script est lancé selon chaque bouton.

3. On programme le fichier "serveur.rb". La première tâche consiste à programmer la fonction "query" utilisée par exemple par "avance.cgi". Elle prend en paramètre une chaîne de caractère et la transmet à un client sur le port 2211. On pourra prendre le fichier "simulationClient.rb" pour le client.

4. On va maintenant modifier "simulationClient.rb" pour transférer le message reçu au serveur TCP lancé sur l'ESP8266. Avec l'aide du second binôme, lancez la commande "AT" pour vérifier que le module répond bien "OK".

5. Depuis la documentation fournie, lancez un serveur TCP sur l'ESP8266 sur le port 8266, puis écrivez un programme "clientTCP.rb" qui se connecte sur ce serveur, et envoie "hello world!". Ce message doit ensuite apparaître dans le Serial monitor.

6. Dans "tunnel.rb", programmez une séquence d'initialisation pour se connecter au serveur TCP, puis une boucle qui : 1) si un message est présent sur le port 2211, le transfère au module ESP8266, 2) si un message est disponible sur le port 8266, l'affiche dans la sortie. Vous veillerez à n'utiliser que des communications non bloquantes.

7. En utilisant les instructions données ici http://codeincomplete.com/posts/ruby-daemons/, créer une classe Tunnel qui peut être démonisée, et qui réalise les mêmes fonctions que tunnel.rb. Le tunnel est lancé depuis initialisation.cgi.

5 Second binôme

5.1 Objectifs

Dans un premier temps, on mettra en route la carte ESP8266 à partir du script "communication.ino" pour dialoguer depuis le Serial monitor de l'IDE Arduino. Par la suite, on réalisera un code permettant de contrôler le moteur pas-à-pas selon les ordres donnés par Internet.

5.2 Procédure

1. Réalisez un schéma de câblage du module en s'aidant de la documentation ESP8266ModuleV1.pdf. Vous ferez attention aux niveaux de tension. Le module ESP8266 peut être alimenté directement par la carte Arduino, mais il sera sous-alimenté lors de l'utilisation du moteur pas-à-pas. Vous alimenterez donc le module par un générateur de tension.

2. En utilisant le script "communication.ino", observez la communication avec le module. Lancez la commande "AT" puis vérifiez que la réponse soit bien "OK".

3. À partir des informations dans la documentation, connectez le module à un réseau WiFi. En l'absence d'un réseau WiFi disponible, vous pourrez lancer un réseau depuis le module, puis connectez votre ordinateur sur ce réseau.

4. Avec l'aide du premier binôme, lancez un serveur TCP sur le module.

5. Déconnectez le module ESP8266, puis connectez un moteur pas-à-pas. Contrôlez le moteur depuis Arduino pour le faire tourner dans un sens ou dans un autre.

6. Rédigez un nouveau programme Arduino qui gère à la fois le module ESP8266 et le moteur pas-à-pas. Vous veillerez à construire des fonctions non bloquantes dans des fichiers externes.

7. Une fois la procédure finie, vous pouvez également aller plus loin, par exemple en mettant à jour le firmware (une commande AT existe pour cela).