PROJET ROBOT EXPLORATEUR

« Cours Systèmes Embarqués »

Table des matières

1. Mission du robot

À la suite d'une défaillance sur un site nucléaire devenu dangereux, on souhaite envoyer un robot d'exploration sur place pour analyser la situation…

Les coordonnées *(longitude, latitude)* du site sont connues. Le robot, doté d'un système GPS, devra s'y rendre de manière autonome. Une fois sur place, le robot *(embarquant une caméra)* sera piloté à distance par un opérateur, qui pourra évaluer la situation grâce aux images transmises.

A noter : *Les informations dont vous avez besoin pour démarrer le projet sont fournies dans ce document.*

2. Travail demandé

Vous devez implémenter les fonctionnalités nécessaires à la mission décrite ci-dessus. Cependant, l'un des objectifs attendus dans ce projet, est de pouvoir joindre votre travail à celui de vos camarades qui travaillent sur tablettes Android ou smartphones.

L'application embarquée dans le robot que vous devez concevoir doit permettre *(entre autres)* à l'application Android *(via le réseau WIFI EMA)* :

- La prise de contrôle du robot
  - Pilotage de tous les actionneurs du robot *(moteurs, pinces, …)*.
  - Lecture des données capteurs *(détecteurs de proximité, capteurs US, Boussole, GPS, état batterie)*.
- L'affichage vidéo en « temps réel » sur la tablette.

Une séance en fin de projet sera réservée aux tests « tablette + robot ».

La navigation du robot pour atteindre le site nucléaire fait également partie des objectifs de ce projet...

2.1. Contraintes

- Le langage de programmation utilisé sera JAVA.
- Toutes les communications *(en phase d'exploitation)* entre le robot et le poste de l'opérateur *(la tablette)* se feront via internet par WIFI. Vous utiliserez les Sockets java pour implémenter les échanges dont le protocole est défini plus bas dans ce document.
- Pour l'affichage vidéo, une image mise à jour toutes les ½s max. devra être accessible sur le serveur WEB Apache installé sur RBPI. Le poste opérateur pourra s'y connecter pour afficher les images…
- L'application ne doit accepter qu'une seule connexion cliente à la fois. Elle doit aussi gérer la déconnexion du client pour permettre une éventuelle nouvelle connexion.
- Pour informer l'opérateur que le robot est prêt *(serveur en attente d'une connexion),* la caméra doit se positionner à droite, puis revenir au centre.

3. Outils de travail

Les outils de développement ne sont pas imposés à l'exception du langage JAVA et de l'API PI4J.

Ci-dessous quelques suggestions :

- **Kate** : pour l'édition du code java *(c'est l'équivalent à NotePad++...)*.
- **L'IDE Eclipse**
- **L'IDE Visual Studio Code**. Un document de mise en œuvre « `MiseEnOeuvre_VisualStudioCode.pdf` » est fourni.

Les outils et ressources déjà installés, ou installés durant les TPs sur la PI4 sont :

✓ **SSH** : Installé en même temps que l'OS. Permet pour la prise contrôle de la PI4 depuis un PC distant.

✓ **Le JDK, ressources JAVA** : Installés sur le robot, par les étudiants durant la phase de TP.

3.1. Configuration réseau

Durant les phases de développement et de mise au point de votre application, utilisez de préférence l'interface Ethernet *(filaire)*, pour vous affranchir d'éventuelles possibles pertes de connexion qui pourraient survenir en WIFI. Le PC a 2 interfaces Ethernet. L'une est connectée au réseau de l'école *(en DHCP)* et donne accès à internet, l'autre a été ajoutée pour les besoins du projet. Cette dernière et celle du robot sont configurées et ont respectivement pour adresse IP : 192.168.0.1 et 192.168.0.2 *(les 2 systèmes forment un réseau local)*.

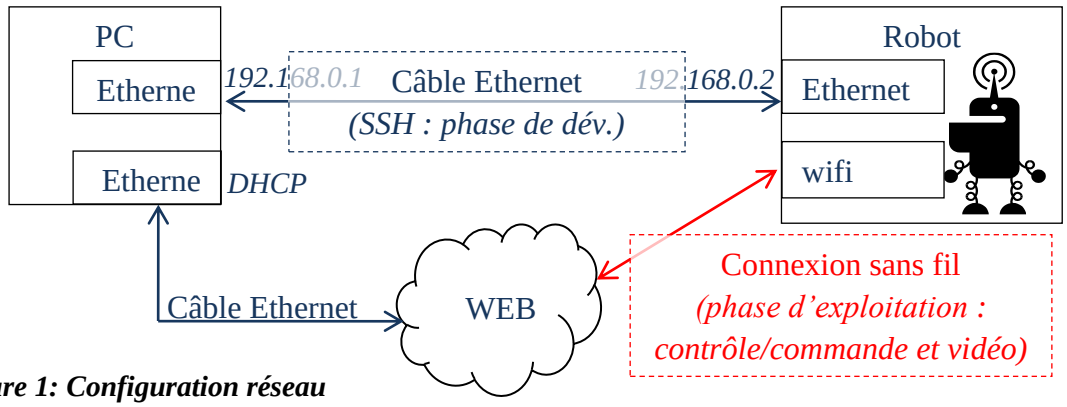
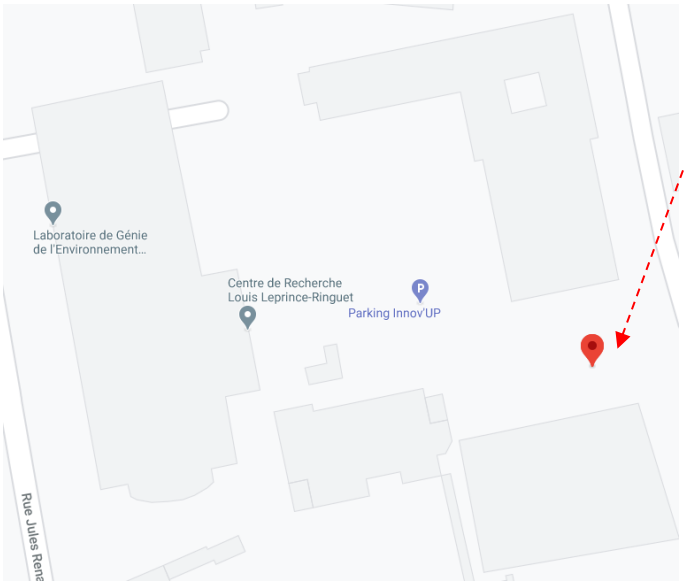


Figure 1: Configuration réseau

A noter : *L'interface WIFI du robot sera paramétrée pour se connecter à un point d'accès de l'école (Mode WIFI managed). Elle donne au robot l'accès à internet. On aurait pu la configurer pour former un réseau local avec le PC (Mode WIFI ad-hoc. Pas de point d'accès). Dans ce cas, il faudrait ploguer sur le PC un dongle WIFI, et le configurer lui aussi en mode ad-hoc.*

Lorsque robot est alimenté par la batterie LIPO 11.1V. Vérifiez régulièrement l'état de charge de la batterie. Il est important de ne pas descendre en dessous de la tension critique de 9.5V (sous peine de destruction de la batterie). La LED rouge « BAT » sur la carte PIC s'allume si la tension est basse. ***A SURVEILLER !!!***

4. Coordonnées du site cible



Latitude : 44°08'10.7"N
Longitude : 4°05'53.6"E

- Obtenues à partir de Google maps *(https://www.google.fr/maps)*.

5. Communication avec le poste opérateur

Le robot doit être capable de répondre à des ordres provenant du poste opérateur distant, via le réseau wifi de l'EMA, qui a été configuré sur le robot durant la phase de TP. Pour cela, vous devez utiliser des Sockets JAVA.

5.1. Les sockets *(généralités)*

Les sockets sont des ressources informatiques qui permettent à 2 machines, l'une cliente et l'autre serveur, de communiquer *(gestion des flux de données entrants et sortants).* Pour cela, les 2 machines doivent avoir une adresse IP et un n° de port.

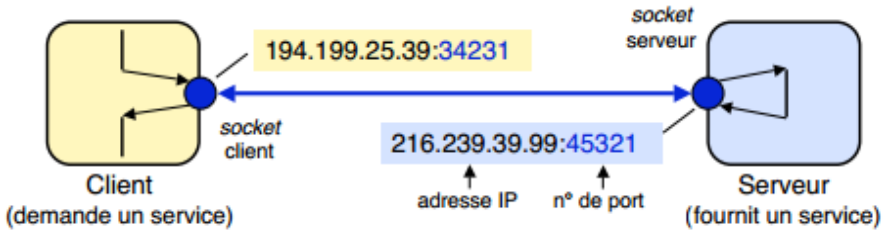


Figure 2: Connexion par socket

L'adresse IP identifie la machine. Le n° port identifie l'application *(le service)* que l'on cible. Cela permet sur une même machine d'avoir plusieurs services qui tournent, par exemple : un serveur WEB et un serveur Socket.

Les numéros de port inférieur à 1024 sont réservés. Ex. de n° de ports connus :

21 : serveur FTP *(File Transport Protocol)*
22 : serveur SSH *(Secure Shell)*
25 : serveur SMTP *(Simple Mail Transfer Protocol*)
80 : serveur HTTP *(HyperText Transfer Protocol)*

2 principaux types de socket :

1. Les sockets basés sur TCP *(Transport Control Protocol)*

   TCP est un protocole de transport en mode connecté. Vous ne pouvez communiquer qu'une fois la connexion établie entre les 2 machines. Ce mode est comparable à une communication téléphonique.

2. Les sockets basés sur UDP *(User Datagram Protocol)*

   Le protocole UDP permet à une machine d'émettre des données vers une autre machine, sans établir de connexion préalable *(la machine réceptrice n'est pas prévenue)*. Le récepteur n'envoie pas d'accusé de réception. Ce mode est comparable à une communication par courrier.

   (De nombreux sites web existent sur le sujet...)

5.2. Les sockets JAVA

Les ressources importantes pour l'utilisation des sockets java se trouvent dans le package `java.net`. Les 2 classes principales qui vont être utilisées sont : les classes `ServerSocket` et `Socket`. Ces ressources java permettent de travailler en mode connecté *(TCP)*.

5.2.1. Diagramme de mise en œuvre *(mode connecté)*

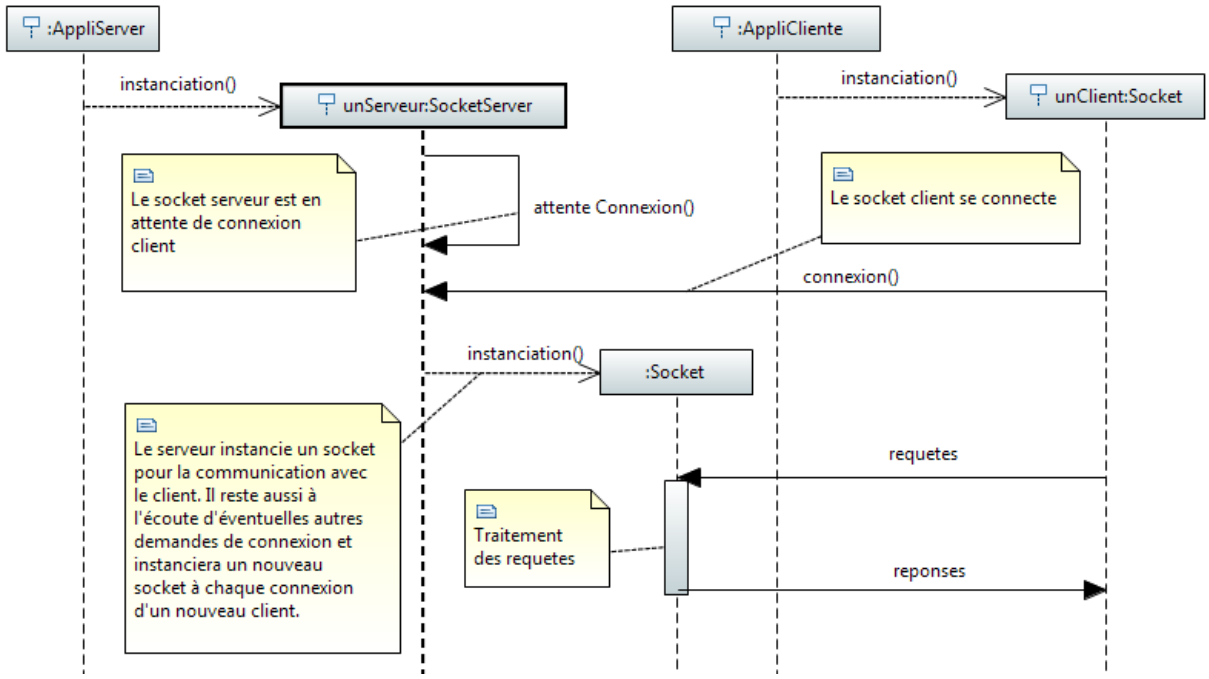


Figure 3: Diagramme de séquence UML : « principe de mise en œuvre des sockets »

Voir aussi : http://docs.oracle.com/javase/8/docs/technotes/guides/net/overview/overview.html

5.2.2. Exemple de mise en œuvre

Un processus serveur est à l'écoute d'un processus client. Si le serveur reçoit la chaine de caractère « `Bonjour !` », il répond « `Salut !` » au client, puis ferme la connexion, sinon il se remet en attente d'un nouveau message ...

Accès aux fichiers → `ServeurSocket.java` / `ClientSocket.java`

```
import java.io.*;
import java.net.*;

public class ServeurSocket
 {
  public static void main(String[] args) throws Exception
  {
   ServerSocket server;
   Socket socket;
   BufferedReader br;//br permet de lire les données envoyées par le client.
   PrintWriter pw; //pw permet d'envoyer des données au client.
   String messageRecu;

   server = new ServerSocket(2009,1,InetAddress.getByName("192.168.0.1"));
   socket = server.accept(); //attente connexion client.
   //Retourne un socket qui va permettre de lire les messages (émis par le
   //client) et d'envoyer des messages vers le client.

   //br permet de lire le flux de données envoyé par le client.
   br = new BufferedReader(new InputStreamReader(socket.getInputStream()));

  //pw permet d'envoyer des données au client.
  //true indique que la vidange du buffer d'émission est automatique.
   pw = new PrintWriter(socket.getOutputStream(), true);

   while(true)
    {
     messageRecu = br.readLine(); // lecture de la trame reçue (bloquant)
     System.out.println("\nMR: "+ messageRecu);
     if (messageRecu.equals("bonjour !"))
      {
       pw.println("Salut !");  //envoie de la trame de réponse
       break; //Interrompt la boucle
      }//fin if
    }//fin while

   br.close(); //fermeture du reader
   pw.close(); //fermeture du writer
   socket.close(); //fermeture de la socket

 }// fin main()
```

```
import java.io.*;
import java.net.*;

public class ClientSocket {

   public static void main(String[] args) throws Exception
   {
    String message="";

    Socket socket = new Socket(InetAddress.getByName("192.168.0.1"),2009);  //Créée un socket client et
                                                                             // se connecte au serveur à l'adresse et N° de port spécifié

    BufferedReader br;//br permet de lire les données retournées par le serveur.
    PrintWriter pw; //pw permet d'envoyer des données au serveur.

    System.out.println("SOCKET = " + socket);

    br = new BufferedReader(new InputStreamReader(socket.getInputStream()));
    pw = new PrintWriter(socket.getOutputStream(),true);

    System.out.println("envoie de bonjour...");
    pw.println("bonjour !");

    message = br.readLine();
    System.out.println(message);
    //On aurait pu ecrire:
    //System.out.println(br.readline()); --> elimine le besoin de la var. message.

    br.close();
    pw.close();
    socket.close();

   }
}
```

5.3. Communication entre la tablette et le robot

L'application JAVA du robot doit traiter les trames entrantes *(requêtes clientes)*, lancer les actions demandées *(par le client)* et retourner des trames de réponse si des données sont demandées. Les messages échangés via le réseau, entre le client *(tablette/smartphone)* et le robot sont des chaines de caractères et doivent respecter un certain formalisme.

Rappel : *la communication entre robot et la tablette s'appuie sur les sockets.*

Il y a 2 types de requêtes envoyées par la tablette :

1. Les requêtes « capteurs » : trames de demande de lecture capteurs avec trames de réponse des données demandées ou trames de non-acquittement *(si la requête est non valide)*.
2. Les requêtes « actions » : trames de « pilotage » du robot avec trames d'acquittement *(acknowledgment)* ou de non-acquittement *(si la requête est non valide)*.

Format des échanges entre la tablette et le robot :

Requête *(envoyée par la tablette)* :

"<nom de la commande>[<espace><paramètre>]<\n>".
([…] : Toutes les requêtes ont 0 ou 1 paramètre)

Trame de réponse *(envoyée par le robot, si la requête est valide)* :

"<Nom de la commande><espace><données demandées séparées par un espace><\n>" ou "<Nom de la commande><espace><donnée demandée><\n>".

Trame accusé de réception *(envoyée par le robot)* :

→ Si la trame n'est pas valide : "<Nom de la commande><espace>TRAME_NOK\n"
→ Si la trame est valide :" <Nom de la commande><espace>TRAME_OK\n"

Quel que soit le type de requête, « actions » ou « capteurs », émise par la tablette, si celle-ci est non valide, le robot doit répondre par un accusé de réception "<Nom de la commande><espace>TRAME_NOK\n". La requête est ignorée par le robot. Dans le cas d'une requête « actions » valide, le robot doit répondre par un accusé de réception "<Nom de la commande><espace>TRAME_OK\n". Dans le cas d'une requête « capteurs » valide, ce sont les data retournées qui font office d'accusé de réception.

A noter : *Toutes les trames se terminent par le caractère '\n'. Une trame est considérée non valide, si elle ne respecte pas le format demandé (nom des commandes, format des paramètres, valeur des paramètres…).*

Quelques exemples :

+ **Requête tablette « *Demande au robot d'avancer à sa vitesse max.* »** : "AVANCER 100\n"
+ Accusé de réception robot *(envoyé à la tablette)* : "AVANCER TRAME_OK\n"
+ Le robot lance l'action demandée…

+ **Requête tablette *(non valide, paramètre manquant !)* « *Demande au robot de reculer à sa vitesse max.* »** : "RECULER\n"

+ Accusé de réception robot *(envoyé à la tablette)* : `"RECULER TRAME_NOK\n"`
+ La requête tablette est ignorée par le robot.

+ **Requête tablette « *lecture des détecteurs de proximité* »** : `"DETECT\n"`
+ Exécute la requête...
+ Réponse robot « *capteurs avant gauche et droit activés* » : `"DETECT AVG AVD\n"`

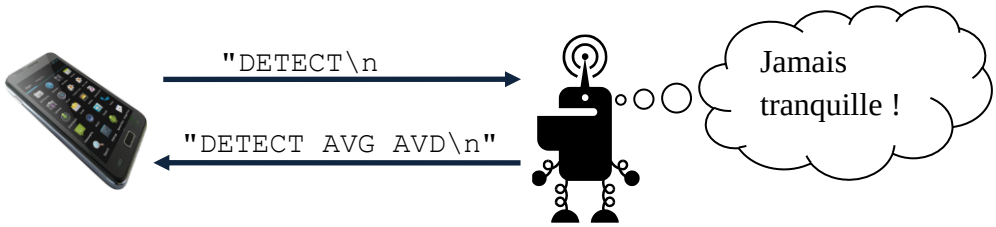


Figure 4: Requête « demande état des détecteurs de proximité »

+ **Requête tablette *(non valide, nom de la commande erroné !)* « *Demande de cap boussole* »** : `"BUSSOLE\n"`
+ Accusé de réception robot *(envoyé à la tablette)* : `"BUSSOLE TRAME_NOK\n"`
+ La requête tablette est ignorée par le robot.

A noter et ne pas oublier :

- *L'application doit gérer la déconnexion d'un client. Si celui-ci se déconnecte, elle doit se remettre en attente d'une nouvelle connexion client.*
- *Pour informer l'opérateur que le robot est prêt (serveur en attente d'une connexion) la caméra doit se positionner à droite, puis revenir au centre.*

5.3.1. Description des requêtes « actions »

Trame émise par la tablette	**Descriptions** *(Actions demandées au robot)*
`"APPLI_STOP\n"`	Demande l'arrêt de l'appli. robot, pour une mise hors tension.
`"AVANCER vit\n"`	Robot avance à la vitesse vit. vit : vitesse exprimée en % de la vitesse max. *(valeurs entières de 0 à 100)*
`"RECULER vit\n"`	Recule à la vitesse vit.
`"GAUCHE vit\n"`	Tourne à gauche à la vitesse vit.
`"DROITE vit\n"`	Tourne à droite à la vitesse vit.
`"ARRET\n"`	Robot stop *(arrêt des moteurs)*.
`"LEVER_PINCE\n"`	Lever la pince.
`"BAISSER_PINCE\n"`	Baisser la pince.
`"OUVRIR_PINCE\n"`	Ouvrir la pince.
`"FERMER_PINCE\n"`	Fermer la pince.
`"VIDEO_START\n"`	Démarrer la prise d'images.
`"VIDEO_STOP\n"`	Stopper la prise d'images.
`"CAM_GAUCHE\n"`	Tourner la caméra à gauche *(du point de vue du robot)*.
`"CAM_DEVANT\n"`	Positionner la caméra devant *(du point de vue du robot)*.
`"CAM_DROITE\n"`	Tourner la caméra à droite *(du point de vue du robot)*.

5.3.2. Description des requêtes « capteurs »

Trame émise par la tablette	Descriptions	Ex. de trames de réponse du robot	Descriptions
`"DETECT\n"`	Demande l'état des détecteurs de proximité. Il y en a 4 : avant droit et gauche (AVD, AVG), arrière droit et gauche (ARD, ARG)	`"DETECT AVG AVD ARG ARD\n"`	Tous les capteurs sont activés. Activé = obstacle. Ordre des data sans importance.
		`"DETECT ARG\n"`	Capteur arrière gauche activé.
		`"DETECT AVG ARD\n"`	Avant gauche et arrière droit activés.
		`"DETECT\n"`	Aucun obstacle
`"BOUSSOLE\n"`	Demande le cap boussole.	`"BOUSSOLE cap\n"`	Cap : angle par rapport au nord en degré. *(valeur entière comprise entre 0 et 360).*
`"DIST_US\n"`	Demande la mesure sur les 3 capteurs US ***(1)***	`"DIST_US distG distC distD\n"`	Distances : gauche, centre et droit en dm. *(valeurs entières comprises entre 0 et 50).*
`"POS_GPS\n"`	Demande de la position GPS	`"POS_GPS lat long\n"`	Latitude et longitude dont le format est : degré°min'sec''N/S ou E/W***(2)***. *(Ex : 44°08'11.2''N)*
`"NBSAT_GPS"`	Demande du nombre de satellites détectés	`"NBSAT_GPS val\n"`	`val` : nombre de satellites *(valeur entière)*
`"ETAT_BATT\n"`	Demande de la tension batterie***(3)***	`"ETAT_BATT V\n"`	V : tension en volt de la batterie. *(valeur réelle, au dixième).*

(1) Fréquence max. de la requête : 500ms.

(2) E/W : indicateur East ou West. N/S : indicateur North ou South

(3) La batterie est considérée complétement déchargée si sa tension est à 9V (évitons le tout de même !), et chargée si sa tension est à 12.6V.

A noter :

- *Pour utiliser le caractère '°', utiliser le codage suivant '\u00B0'. Pour insérer des guillemets doubles (caractère : '"') dans une chaine utiliser le codage '\"'.*
  `Ex : System.out.println("Latt.: 44\u00B008'11.2\" N");//44°08'11.2" N.`

6. Architecture fonctionnelle de l'application sur le robot

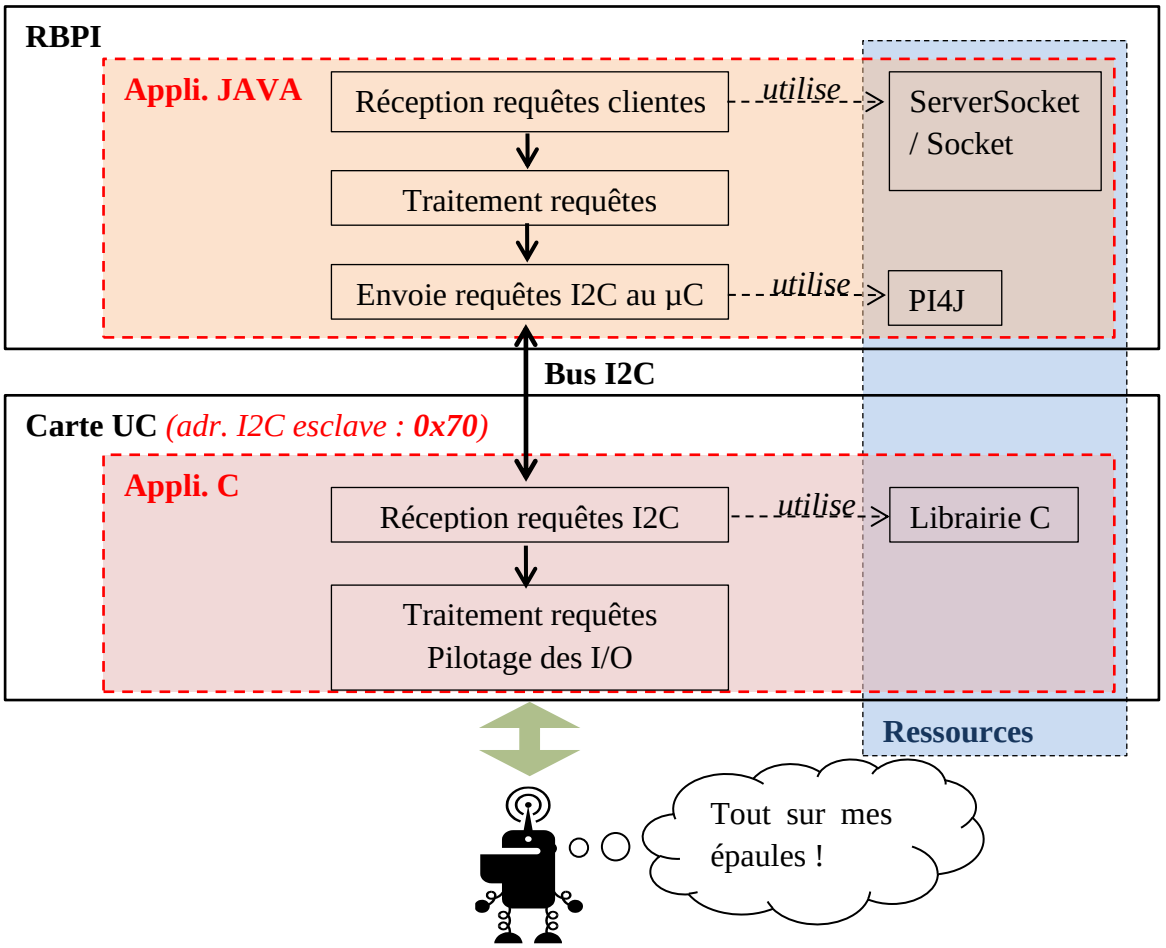


Figure 5: Architecture fonctionnelle de l'application sur le robot

7. Fonctions implémentées sur la carte µC du robot

Une grande partie des capteurs et actionneurs du robot *(excepté la caméra, le GPS, l'interface WIFI, et le module RF433)* sont gérés par la carte µC PIC18F46K22 connectée au RBPI par un bus I2C. Le programme C sur la carte µC attend des octets de commande envoyés par le RBPI via ce bus, et les exécute.

Toutes les commandes ont le format suivant : `<1 octet de commande><1 octet paramètre>`. Le paramètre n'est pas toujours utilisé.

Ci-dessous, la liste des commandes implémentées dans le programme de la carte µC PIC PIC18F46K22, que le RPI peut invoquer.

7.1. Contrôle du programme *(commande « STOP »)*

Cette commande permet de « stopper » le programme µC. Au démarrage, le programme µC initialise les périphériques du PIC, puis démarre le service d'écoute et d'exécution des requêtes `I2C` *(qui seront émises par le RBPI)*. La commande « `STOP` », lorsqu'elle est invoquée, stoppe l'écoute des requêtes I2C et lance une boucle infinie faisant clignoter la `LED_INFO_UC` *(LED jaune sur le `PORT RC5`)*. Ceci dans l'attente d'une mise hors tension du système. Le paramètre est sans effet, vous pouvez passer 0 par exemple.

- `CMD_CPU_STOP :` `0x00 0x00`

7.2. Contrôle des moteurs

Ces commandes permettent de contrôler le déplacement du robot. Un paramètre de vitesse peut être fixé. Nous avons vu dans le cours microcontrôleur, que le paramètre de vitesse max. était 333 *(valeur max. à passer à la fonction permettant de fixer le rapport cyclique du PWM, de manière à obtenir la vitesse max.)*. La valeur 333 est codée sur 2 octets. Hors un seul octet est passé en paramètre *(valeur comprise entre 0 et 255)*. Le programme PIC ajoute la valeur 78. Ce qui signifie que la vitesse n'est jamais nulle. *(Ex. : si le paramètre passé est 0, la valeur fixant le rapport cyclique sera de 78 (0 + 78).*

- `CMD_ROBOT_AVANCER :` `0x10 <VITESSE>`
- `CMD_ROBOT_STOP :` `0x11 <SANS_EFFET (passer 0 par ex.)>`
- `CMD_ROBOT_TOURNER_D :` `0x12 <VITESSE>`
- `CMD_ROBOT_TOURNER_G :` `0x13 <VITESSE>`
- `CMD_ROBOT_RECULER :` `0x14 <VITESSE>`

7.3. Contrôle de la pince

Ces commandes permettent le contrôle de la pince *(pilotage des 2 servomoteurs)*. Le paramètre est sans effet, passer 0.

- `CMD_PINCE_OUVRIR :` `0x20 0x00`
- `CMD_PINCE_FERMER :` `0x21 0x00`
- `CMD_PINCE_LEVER :` `0x22 0x00`
- `CMD_PINCE_BAISSER :` `0x23 0x00`

7.4. Contrôle moteur caméra

Cette commande pilote le servomoteur de la caméra *(le PAN)*. Le paramètre est sans effet, passer 0.

- `CMD_CAM_TOURNER_G :` `0x30 0x00`
- `CMD_CAM_CENTRER :` `0x32 0x00`
- `CMD_CAM_TOURNER_D :` `0x31 0x00`

7.5. Contrôle capteurs US

Cette commande retourne la mesure de distance effectuée sur les 3 capteurs US. Le paramètre est sans effet, passer 0. Elle retourne 3 octets.

- 1er octet : mesure distance en dm sur capteur US gauche.
- 2ième octet : mesure distance en dm sur capteur US centre.
- 3ième octet : mesure distance en dm sur capteur US droit.

Le temps d'une mesure US sur les 3 capteurs est de ~500ms *(oui c'est assez lent ! Ça pourrait sans aucun doute être amélioré…)*. Ce qui signifie que le RPI doit invoquer une lecture des mesures US 500ms *(minimum)* après la requête de demande de mesure.

- `CMD_CAPT_US_DIST :  0x40 0x00`

- **Les étapes de mise en œuvre :**

1. Le RPI envoie la requête I2C « `<0x40><0x00>` » de demande de mesure US à la carte PIC.
2. La carte PIC exécute la mesure sur les 3 capteurs US *(~500ms) (|| le RPI attend)*.
3. Le RPI exécute une lecture I2C sur la carte PIC pour récupérer les mesures…

7.6. Contrôle capteur IR

Cette commande retourne l'état des capteurs IR. Le paramètre est sans effet, passer 0. Le programme du PIC retourne la donnée codée sur un octet, de la manière suivante :

- `0000<IR_AVG><IR_AVD><IR_ARG><IR_ARD>`

- `CMD_IR_DETECT :  0x50 0x00`

- **Les étapes de mise en œuvre :**

1. Le RPI envoie la requête I2C « `<0x50><0x00>` » de demande de lecture détecteurs IR à la carte PIC.
2. La carte PIC lit l'état des détecteurs IR *(<<1ms) (|| le RPI attend)*.
3. Le RPI exécute une lecture I2C sur la carte PIC, pour récupérer l'état des détecteurs…

7.7. Contrôle boussole

Cette commande retourne le cap sur 2 octets. Le format est le même que celui retourné par la boussole *(voir doc. HMC6352.pdf)*. Le paramètre est sans effet, passer 0.

- `CMD_BOUSSOLE :  0x60 0x00`

- **Les étapes de mise en œuvre :**

1. Le RPI envoie la requête I2C « `<0x60><0x00>` » de demande de CAP à la carte PIC.
2. La carte PIC fait l'acquisition du CAP *(~20ms) (|| le RPI attend)*.
3. Le RPI exécute une lecture I2C sur la carte PIC pour récupérer l'octet de poids fort du cap.
4. Le RPI exécute une lecture I2C sur la carte PIC pour récupérer l'octet de poids faible du cap.

7.8. Contrôle batterie

Cette commande permet de retourner la valeur numérique de conversion *(comprise entre 0 et 1023 (10bits))* de la tension batterie sur 2 octets. Le premier octet retourné est le poids fort, ensuite vient le poids faible. Pour calculer la tension, on sait que :

- ✓ `VN = 0` → `Vbat=0V`
- ✓ `VN = 1023` → `Vbat ~12.6V`

- `CMD_BATT :  0x70 0x00`

- **Les étapes de mise en œuvre :**

1. Le RBPI envoie la requête I2C « `<0x70><0x00>` » de demande tension batterie à la carte PIC *(le paramètre est sans effet)*.
4. La carte PIC fait l'acquisition de la tension *(<<1ms) (|| le RPI attend)*.
2. Le RBPI exécute une lecture I2C sur la carte PIC pour récupérer le poids fort de la valeur numérique, image de la tension.
3. Le RBPI exécute une lecture I2C sur la carte PIC pour récupérer le poids faible de la valeur numérique, image de la tension.

8. Mise en œuvre du bus I2C avec PI4J

Tout d'abord, il faut savoir que le RBPI est doté de 2 bus I2C *(le bus 0 et le bus 1)*. Celui que nous mettons en œuvre est le bus 1, disponible sur les broches 3 et 5 du connecteur P1 *(voir cours…)*. Le bus I2C sur le RBPI, ne peut être qu'en mode MAITRE. Ça tombe bien ! La carte PIC qui est connectée sur le bus 1, est en mode ESCLAVE. Ce qui signifie, pour rappel, que c'est le RBPI qui initie la communication, la carte PIC exécute.

Pour la mise en œuvre java, vous devez utiliser les ressources disponibles dans le package *com.pi4j.io.i2c (voir https://pi4j.com/1.4/apidocs/index.html)*.

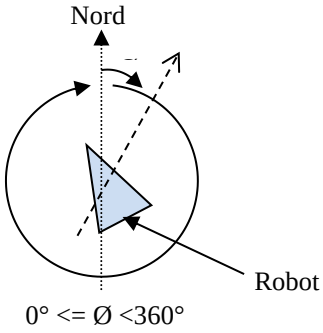
Notamment :

- La classe **I2CFactory :** permet de créer les objets qui vont vous donner l'accès au bus I2C, comme vous l'avez fait pour les I/O tout ou rien.
- La classe **I2CBus** : permet de sélectionner, grâce à la méthode `getDevice()`, un système cible connecté à notre bus I2C *(ici c'est la carte PIC. A noter, qu'elle a pour adresse **0x70**)*.
- L'interface **I2CDevice** : permet d'écrire/lire des données en direction du device cible esclave sélectionné.

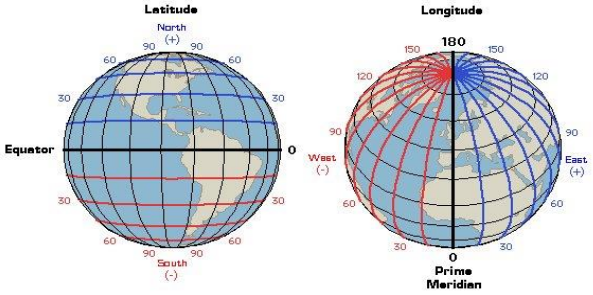
9. Positionnement GPS

Utilisation du récepteur GPS *(référence : SUP500R)*. Il est connecté sur le port série UART0 présent sur le connecteur P1 *(broches 8 et 10)*. La documentation du module est fournie.

Cap boussole



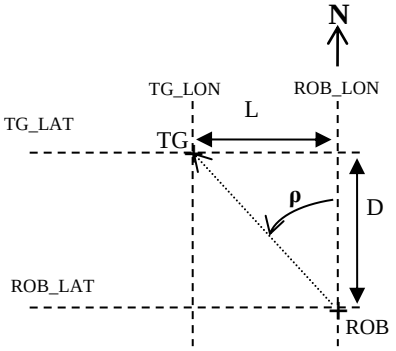
Longitude et latitude



La longitude : comprise entre -180° *(vers l'ouest)* et +180° *(vers l'est)*.
La latitude : comprise entre -90° *(sud)* et +90° *(nord)*.

Approximations

On considère la terre ronde, et compte tenu du rayon de la terre et des faibles distances entre la cible et le robot *(on néglige la courbure)*, on pose :




```
L = R*(ROB_LON - TG_LON)
D = R*(TG_LAT - ROB_LAT)
```

```
ρ = arctan(L/D) +/- nπ = arctan((ROB_LON-TG_LON)/(TG_LAT-ROB_LAT)) +/- nπ
```

`ROB_LON` et `TG_LON` : respectivement longitude du robot et de la cible.
`ROB_LAT` et `TG_LAT` : respectivement latitude du drone et de la cible.
`R` : rayon de la terre = `6371km`
(ROB : Robot / TG : Target)

Nord Magnétique et Nord géographique sont considérés identiques.

Pour calculer la distance entre 2 points GPS A et B, on trouve la formule suivante … : *(http://geodesie.ign.fr/contenu/fichiers/Distance_longitude_latitude.pdf)*

```
Dist = R*arcos[sin(latA)*sin(latB) + cos(latA)*cos(latB)*cos(longB-longA)]
```

9.1. Mise en œuvre du port série avec PI4J

Voir un exemple en ligne sur le site PI4J *(https://www.pi4j.com/1.4/example/serial.html)*.

Pour la configuration du port série sur lequel est connecté le GPS, vous devez consulter la doc. du GPS: « *SUP500R_v5.pdf* ».

Important : n'oubliez pas que dans notre cas, nous n'utilisons pas l'UART en écriture. Nous cherchons juste à traiter les trames provenant du GPS *(pas de* `serial.write(…);` *pour nous…)*.

Toutes les infos dont vous avez besoin pour comprendre l'exemple et aller plus loin se trouvent en ligne : https://www.pi4j.com/1.4/

10. Gestion de la vidéo

La caméra est connectée sur le RBPI par un connecteur dédié *(Interface CSI)*. La vidéo fournie par le RPI, doit être accessible par connexion WIFI *(dongle WIFI plogué sur l'un des 2 ports USB disponibles sur la RPI)*.

Les applications `libcamera-vid` et `libcamera-still` *(utilisée dans notre cas)* permettent respectivement de générer de la vidéo et des images. On trouve de la doc. sur internet…

10.1. Mise en œuvre en JAVA

Pour lancer depuis JAVA l'application `raspistill`, nous allons utiliser la classe `Runtime` *(dans le package* `java.lang`*)*. Elle permet de lancer des commandes Shell.

Exemple :

```
Runtime.getRuntime().exec("mkdir /home/pi/nouveauRep"); //lance la cmde shell mkdir qui créée un nouveau répertoire « nouveauRep »…
```

A noter : *L'application* `raspistill` *doit être paramétrée pour générer un fichier «* `vueRobot.jpg` *» toutes les 1/2sec max. Cette image doit être déposée dans le répertoire «* `/var/www/html` *». C'est le répertoire dans lequel Apache va chercher les pages WEB à retourner aux internautes …*

11. Lancement automatique de l'application au démarrage de Linux

Ajouter avant l'instruction « `exit(0)` » dans le fichier `/etc/rc.local` :

```
cd /home/pi/DEV/<votre ou se trouve l'appli java>
export CLASSPATH=.:/opt/pi4j-1.4/lib/'*'
java NomDeLappliRobot &
```

12.Bibliographie

http://fr.wikipedia.org/wiki/NMEA_0183
https://openclassrooms.com/fr/courses/6173501-apprenez-a-programmer-en-java
http://www.commentcamarche.net/contents/1054-programmation-reseau-les-sockets
http://www.commentcamarche.net/contents/528-port-ports-tcp-ip