Rapport de UE8
Programmation réseaux

Kelun CHAI
Djaber SOLIMANI
Nogaye GNING

1. Construction du robot

Nous avons choisi un robot à chenilles, il s'agit du robot *RaspTank* de la marque *Adeept*, qui comprend l'ensemble caméra et moteurs, des LEDs, et est alimenté par batterie. Il doit être utilisé avec un Raspberry Pi.

La construction du robot a été prise en charge par Nogaye et nous avons réussi à assembler les pièces selon le manuel.

2. Configuration du Routeur

La configuration du serveur a été faite par trois personnes avec l'aide du professeur. Au début, le fichier de configuration est fourni par Kelun et Nogaye essaye de le configurer sur un RaspBerry Pi. Le logiciel hostapd est installé, nous utilisons aussi dnsmasq pour configurer un serveur DHCP.

```
interface=wlan0
driver=nl80211
ssid=RouteurRasp
hw_mode=g
channel=7
wmm_enabled=0
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=rasp123456
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

Mais après l'avoir configuré, nous avons constaté que nous ne pouvions pas nous connecter à ce hotspot. Nous avons résolu ce problème après avoir redémarré le service.

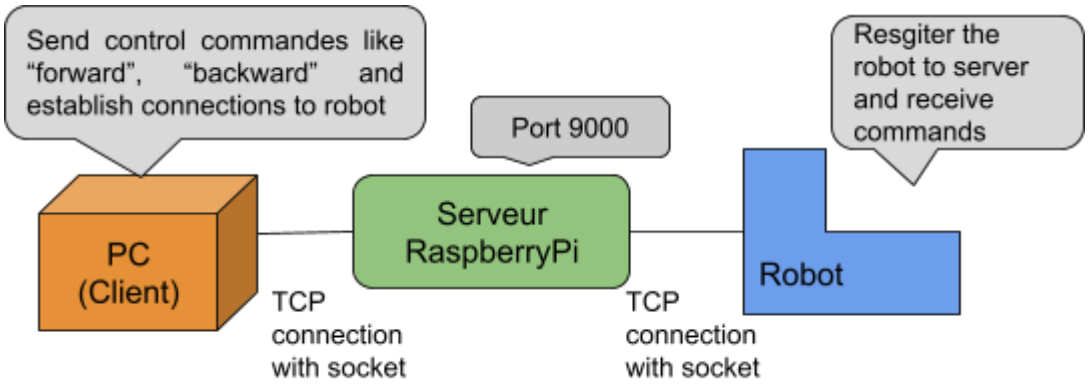
3. Programmation TCP

L'ensemble des programmes est stocké dans ce dépôt git :
https://github.com/klchai/RaspRobot

Il est composé de 3 fichiers python (`client.py, robot.py` et `server.py`), ainsi que d'un README.

a. Serveur & Client

Djaber s'est occupé de toute la partie communication réseau pour implémenter le protocole TCP défini. Il gère aussi le test entre le serveur, le client et le robot.



Il a pour cela utilisé l'API socket de Python afin de faire communiquer ensemble le client (le pc depuis lequel on saisit les commandes), le serveur (le routeur qui fait l'intermédiaire entre le client et le robot) et enfin le robot.
Le code du client est dans le fichier client.py, celui du serveur dans le fichier server.py du dépôt Git.

a.1 Gestion de l'enregistrement

Le nom du robot est stocké dans un dictionnaire. Lors de l'enregistrement d'un nom, si le nom existe déjà, alors une erreur "`already used`" est retournée, sinon le nom est ajouté au dictionnaire.

```
if msg.startswith("register") and len(msg.split()) == 2:
    sockrobot = client
    robot_name = msg.split()[1]
    if robot_name in robots:
        msg = f"Robot {robot_name} already registered"
    else:
        msg = "ok"
        robots[robot_name] = None
        print("add robot ",robot_name)
```

a.2 Gestion de la connexion

La connexion entre le client et le robot est également gérée par le dictionnaire. Si le robot auquel vous souhaitez vous connecter n'existe pas, un message d'erreur "`unknown`" est renvoyé. Si le robot correspondant est déjà lié à un client, un message d'erreur "already used" est renvoyé. En outre, le robot se connecte au premier client qui se connecte à lui.

```
elif msg.startswith("connect") and len(msg.split()) == 2:
    robot_name = msg.split()[1]
    if robot_name not in robots:
        msg = f"Robot {robot_name} unknown"
    elif robots[robot_name] is not None:
        msg = f"Robot {robot_name} already used"
    else:
        robots[robot_name] = client
        msg = "ok"
        print("pc connected to robot ",robot_name)
```

a.3 Threading

Le serveur utilise plusieurs threads, 1 pour chacun de ses clients, on utilise pour cela l'API `threading`.

```
print(f"Server is started at port {PORT}")
try:
    while True:
        client,_ = sock.accept()
        threading.Thread(target=handler, args=(client,)).start()
except KeyboardInterrupt:
    print("Closing server socket...")
    sock.close()
```

b. Robot

La partie robot a été réalisée par Kelun et le test final a été effectué par Djaber. Le robot se connecte au port 9000 du serveur et peut s'enregistrer sur le serveur. Si le nom est utilisé, le serveur retournera le message d'erreur déjà utilisé.

Le Raspberry Pi utilise le PCA9685 pour contrôler les moteurs, qui peuvent émettre des signaux d'impulsion PWM (Pulse Width Modulation) pour la commande. Nous avons également réglé le `GPIO` sur le mode `BCM` pour utiliser la numérotation électrique pour la puce.

```
import Adafruit_PCA9685
import RPi.GPIO as GPIO
```

En se référant au code officiel d'*Adeept*, nous définissons les broches correspondant aux deux moteurs comme broches de sortie.

```
def setup():#Motor initialization
        global pwm_A, pwm_B
        GPIO.setwarnings(False)
        GPIO.setmode(GPIO.BCM)
        GPIO.setup(Motor_A_EN, GPIO.OUT)
        GPIO.setup(Motor_B_EN, GPIO.OUT)
        GPIO.setup(Motor_A_Pin1, GPIO.OUT)
        GPIO.setup(Motor_A_Pin2, GPIO.OUT)
        GPIO.setup(Motor_B_Pin1, GPIO.OUT)
        GPIO.setup(Motor_B_Pin2, GPIO.OUT)
```

b.1 Gestion de la direction

Afin de contrôler la direction, nous devons ajuster la sortie du moteur. En réglant la puissance de sortie de différentes broches, nous pouvons changer la tension correspondante par `GPIO.HIGH` ou `GPIO.LOW`. `HIGH` signifie le démarrage complet, et `LOW` signifie que la tension de sortie est 0.
Pour tourner à gauche, il suffit de démarrer le moteur de droite (Motor 2) et de lui faire suivre un certain arc. Le virage à droite dépend du moteur de l'autre côté.

```
def left(status, direction, speed):#Motor 2 positive and negative rotation
        if status == 0: # stop
                GPIO.output(Motor_B_Pin1, GPIO.LOW)
                GPIO.output(Motor_B_Pin2, GPIO.LOW)
                GPIO.output(Motor_B_EN, GPIO.LOW)
        else:
                if direction == Dir_backward:
                        GPIO.output(Motor_B_Pin1, GPIO.HIGH)
                        GPIO.output(Motor_B_Pin2, GPIO.LOW)
                        pwm_B.start(100)
                        pwm_B.ChangeDutyCycle(speed)
                elif direction == Dir_forward:
                        GPIO.output(Motor_B_Pin1, GPIO.LOW)
                        GPIO.output(Motor_B_Pin2, GPIO.HIGH)
                        pwm_B.start(0)
                        pwm_B.ChangeDutyCycle(speed)
```

Pour avancer et reculer, il faut faire en sorte que les deux moteurs avancent ou reculent en même temps à une certaine vitesse.

```
if direction == 'forward':
        if turn == 'right':
                left(0, left_backward, int(speed*radius))
                right(1, right_forward, speed)
        elif turn == 'left':
                left(1, left_forward, speed)
                right(0, right_backward, int(speed*radius))
        else:
                left(1, left_forward, speed)
                right(1, right_forward, speed)
```

Voici le code pour la marche avant, nous appelons les fonctions left et right pour que les deux moteurs se déplacent dans la direction de la marche avant.

b.2 Gestion de la vitesse

Pour modifier la vitesse du moteur, il faut modifier la fréquence de sortie du signal. On utilise la fonction `pwm.ChangeDutyCycle(int)` pour modifier la fréquence de pulsation. Nous définissons une variable globale pour gérer la vitesse, la valeur initiale est de 60. Tous les mouvements suivront cette vitesse.

```
        elif direction == Dir_backward:
                GPIO.output(Motor_A_Pin1, GPIO.LOW)
                GPIO.output(Motor_A_Pin2, GPIO.HIGH)
                pwm_A.start(0)
                pwm_A.ChangeDutyCycle(speed)
    return direction
```

Cette variable sera modifiée après avoir accepté un message de changement de vitesse de la part du client.

```
elif command.startswith("speed") and len(command.split())==2:
    new_speed = int(command.split()[1])
    init_speed = int(new_speed)
    print(f"Speed now is {init_speed}")
```

4. Examen

Lors de l'examen nous n'avons pas réussi à nous connecter aux serveurs de nos camarades. En effet, nous avons configuré le Raspberry Pi du robot afin qu'il se connecte au réseau "`RouteurRasp`", alors que nos camarades ont utilisé des noms de réseaux différents. Nous aurions dû nous mettre d'accord tous ensemble avant cela. Malgré plusieurs essais, nous avons réussi à connecter notre PC client aux autres réseaux mais pas le Raspberry Pi du robot, qui nécessitait que l'on se connecte à lui via ssh mais nous ne disposions pas d'écran dans le hall (là où l'on devait faire la course des robots) pour pouvoir le configurer.
Nous avons cependant réussi à contrôler notre robot via notre propre serveur, et à le montrer au professeur.