Comment fonctionne le serveur ?

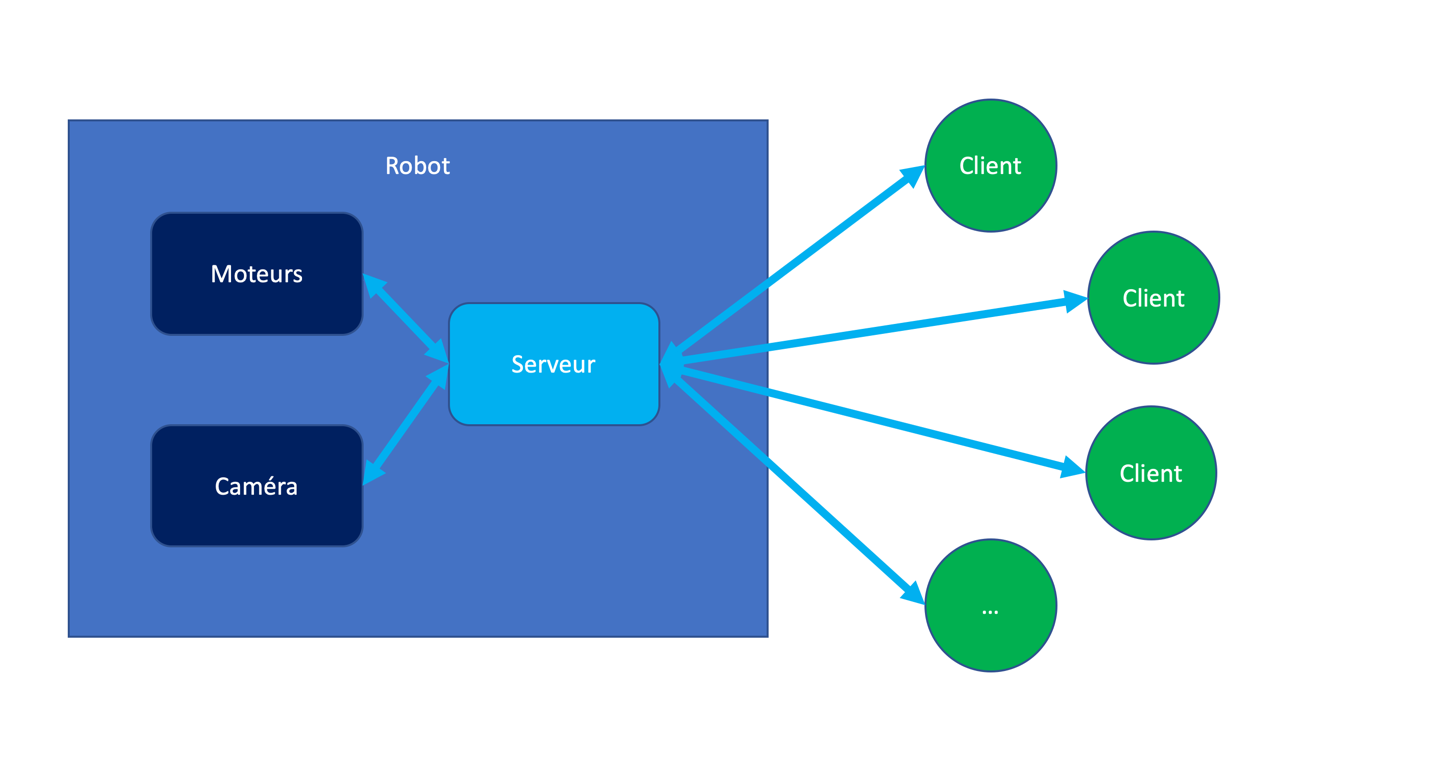
Pour pouvoir contrôler le robot à distance, nous avions besoin de créer un programme permettant de lui envoyer des instructions à distance. C’est à ce moment la que nous ai venu l’idée de le rendre contrôlable de n’importe où à distance à l’aide d’un site web. Mais pour cela, il nous fallait donc un serveur.

Le serveur est donc la partie permettant de lier le robot à n’importe qu’elle autre appareil électronique disposant d’internet et d’un navigateur web (bien sûr pour empêcher tout problèmes de sécurité, nous l’utilisons seulement en local, et non sur internet). Ce serveur est un programme codé en Python (version 3.0 ou supérieur), il est composé en lui-même de deux fichiers :

- main.py : Il contient la méthode principale du programme, qui est appelé dès qu’il est exécuté et va permettre de créer la classe serveur dans un thread (cela permet de continuer d’exécuter le serveur sans interrompre la fonction principale).

- server.py : Il contient la classe du serveur, son fonctionnement est le suivant, lorsqu’il est créé, il va créer un « socket » qui correspond dans la librairie standard python à un serveur à l’aide du protocole TCP/IP qui est un protocole de transport en mode connecté. Ce serveur va ensuite attendre en boucle (d’où l’intérêt du thread pour ne bloquer aucune exécution ou transmission de message que la boucle pourrait causer) qu’un serveur lui envoie une demande de connexion (à l’aide de l’IP de la machine ou est exécuté le serveur). Une fois un client connecté, il pourra envoyer un message au serveur (qui sera compris comme une instruction) et qui sera donc lu par le serveur afin d’effectuer certaines actions directement sur le robot et d’envoyer une réponse au client.

Le serveur est donc directement connecté aux deux classes permettant le fonctionnement des différentes parties du robot (moteurs et caméra) en possédant lui-même des threads de ces deux classes qui tourneront aussi en boucle en attente de donnés ou d’instructions (ex : changer les vitesses pour le moteur, envoyer le flux vidéo pour la caméra).



Maintenant, il faut aussi comprendre comment le serveur et le client arrive à communiquer grâce au protocole TCP/IP. Étant donné que le client est un site web, et non un autre programme en python (même si cela est possible, mais nous avons préféré le site web pour la flexibilité et la personnalisation plus avancé et facilité), le protocole TCP/IP n’est pas directement utilisable, il faut utiliser le protocole WebSocket, qui est un protocole liant le « World Wide Web » à un serveur à l’aide d’une « surcouche » (canaux de communications différents) appliqué sur le protocole TCP/IP.

Tout d’abord, lors d’une tentative de connexion au client, le protocole WebSocket va envoyer des données au serveur le notifiant de cette tentative, ces données de connexion forment ce qui est appelé dans ce protocole, un « handshake » (poignée de main). Ces données contiendront différentes informations, le serveur va donc devoir traiter ces données, et obtenir la « Sec-WebSocket-Key », qui est une clé unique à chaque connexion et contenu dans ces données (mais encodé au format base64). Pour que la connexion soit correctement effectuée, le serveur va donc devoir retourner au client une réponse précise, dans le même format que ce qui était envoyé par le client, mais le point le plus important de cette réponse est le « Sec-WebSocket-Accept » qui est une valeur que le serveur doit créer et qui doit être très précise pour que la connexion soit acceptée. Cette valeur doit contenir la valeur de hachage SHA-1 encodé au format base64 de la clé reçu précedemment, tout d’abord décodé, puis concaténé avec la chaîne de caractère suivante « 258EAFA5- E914-47DA-95CA-C5AB0DC85B11 » (qui forme un identifiant universel ayant un taux de chance très peu élevé d’être reconnu par les autres données transitant sur internet ne passant pas par le protocole WebSocket).

Ensuite, pour pouvoir faire transiter des messages, il faut toujours respecter le protocole WebSocket, pour cela, les messages envoyés doivent être encoder pour respecter un certain format, et ceux reçu doivent être décoder pour être lisible.

Le décodage fonctionne en recevant un message en binaire, on doit d’abord lire la taille du paquet (chiffre contenu sur le second byte) puis selon la taille du paquet, lire la taille du message, suivi de la lecture du « mask » (qui est une sorte de « masque » qui permet de sécuriser un message en le codant) et d’ensuite lire le message sur le nombre de byte donné par la taille du message obtenu précédemment, puis finalement, d’appliquer le « mask » dessus pour le décoder et le rendre compréhensible au format unicode (pour supporter les caractères spéciaux).

L’encodage fonctionne du même principe, mais en inverse, à la différence que l’encodage des données pour les envoyer au client ne contient aucun « mask ».

Le serveur est donc une partie fondamentale du projet, permettant la transmission d’information entre chaque partie du robot et les clients.