Compte rendu TP2 Robotique

Brunner Baptiste
Carrière Lilian
BUT GEII 2 ESE A

Table des matières

1) A la découverte de la programmation orientée objet en C#

La phase actuelle des TP vise à nous familiariser avec la programmation en langage C# et le logiciel Visual Studio. L'objectif principal est de concevoir et de mettre en forme une interface utilisateur permettant une interaction avec le robot programmé au cours de la phase précédente de ce travail pratique. L'objectif final de cette phase est d'établir un contrôle sur le robot, en utilisant soit l'interface développée dans Visual Studio, soit le clavier d'un ordinateur. Pour atteindre ces objectifs, une compréhension approfondie des protocoles de communication entre différentes machines est impérative. Il faut donc être en mesure de mettre en œuvre ces protocoles de manière efficace, assurant ainsi une communication fiable et réactive entre l'interface utilisateur et le robot.

1.1) Simulateur de messagerie instantanée.

La première partie de ces TP permet d'apprendre à utiliser le logiciel Visual Studio et la programmation en C#. Il est d'abord demandé de mettre en place une interface graphique permettant d'envoyer et de recevoir des informations.

Il faut d'abord produire la partie graphique de l'interface. Pour ce faire, rien de compliqué, car Visual Studio comporte un système de « drag and drop » avec un grand nombre d'éléments déjà programmés. Après avoir sélectionné les éléments qui apparaîtront sur l'interface et les avoir placés, il est alors possible de commencer à inclure certaines fonctionnalités. La première de celles-ci est l'envoi du texte situé dans la TextBox d'émission vers la TextBox de réception via l'appui sur la touche entrée ou le bouton « envoyer » de l'interface.

1.2) Messagerie instantanée entre deux PC.

Il s'agit ici de mettre en place les programmes qui permettront d'envoyer des données sur un port série USB depuis un PC vers un autre, ou sur le même PC si l'on utilise un système de loopback. Pour permettre ce mode de communication, il faut alors utiliser un ou deux modules FT232RL (ou un autre module de même type).
Pour pouvoir communiquer via un port série de l'ordinateur, il est possible d'utiliser la librairie ReliableSerialPort sur Visual Studio. Cette dernière permet d'utiliser les ports USB de l'ordinateur et inclut plusieurs fonctions permettant de communiquer grâce à ces ports.
La fonction d'envoi des données de la TextBox d'émission est donc modifiée pour qu'elle puisse envoyer les informations sur le port USB sélectionné. Pour vérifier que le système de communication fonctionne bien, on utilise le loopback afin de simuler l'envoi d'un message par un second ordinateur. On se rend alors compte qu'il faut modifier la façon dont les informations reçues sont traitées si l'on souhaite pouvoir faire fonctionner le système de messagerie instantanée. En effet, il faut prendre en compte le fait que notre programme ne peut pas effectuer plusieurs actions en même temps. Il faut donc stocker les données reçues dans une variable qui ne fait pas partie de la fonction de réception. Ensuite, si l'on souhaite récupérer ces informations pour les utiliser, il faut lire le contenu de la variable à un intervalle régulier. Cela permet de délayer le traitement des données de leur réception, et donc d'éviter de demander au programme de faire deux opérations à la fois. Une fois que ce système de Timer a été mis en place, il est donc possible d'afficher les données qui ont été préalablement transmises par l'ordinateur dans le port USB. Le système de communication entre deux ordinateurs est donc opérationnel.

1.1) Structuration du code à l'aide d'une classe Robot.

Pour faciliter la lecture et la compréhension du programme, le choix est fait de stocker toutes les données en provenance du robot dans une classe « Robot ». Cela permet donc de structurer notre programme en utilisant les propriétés des classes en programmation.
Il faut alors mettre à jour le programme et changer toutes les variables des fonctions de réception pour qu'elles correspondent aux nouvelles variables de la classe « Robot ».

1.2) Liaison série hexadécimale.

Le système de communication qui est actuellement mis en œuvre dans le projet Visual Studio permet de transmettre des chaînes de caractères d'un ordinateur vers un autre rapidement. Cependant, il n'est pas possible de transmettre tous les caractères que l'on pourrait être amené à utiliser. Par soucis de compatibilité et pour permettre des échanges plus précis, il faut donc revoir la manière dont sont codés les messages entre ordinateurs. Afin de pouvoir transmettre les caractères qui seront nécessaires au contrôle du robot, il faut alors changer les chaînes de caractères en chaînes d'octets en hexadécimal. Cette façon de faire permet d'envoyer tout type de données comme des valeurs, des messages, ou des caractères de contrôle.

Pour mettre en place ce système de communication par octets, il est nécessaire de remplacer la variable de stockage de texte par un buffer de type FIFO. En programmation, un buffer correspond à un espace de stockage de données temporaire, et il se remplit caractère par caractère. Son but est de servir d'intermédiaire lors d'une opération de transfert de données par exemple. Il remplit donc parfaitement le rôle de la variable utilisée jusqu'à présent et permet donc de la remplacer.
La spécificité d'un buffer de type FIFO, c'est que comme son nom anglais l'indique, « First In, First Out », il est rempli puis vidé. Cela signifie que lorsqu'on transmet des données dans ce stockage, il est ensuite possible de les récupérer dans l'ordre dans lequel elles ont été envoyées. Dans le cas du projet de messagerie entre deux ordinateurs, ce principe de fonctionnement permet d'envoyer un grand nombre de données, puis de les stocker afin de pouvoir les analyser lorsque le programme aura le temps. Cela permet d'éviter de bloquer les autres processus lors de la réception d'informations.
Pour être en mesure d'afficher ces données et qu'elles soient lisibles pour un utilisateur humain, il faut penser à convertir les octets reçus en chaîne de caractères. La fonction ToString permet de réaliser cette conversion et permet aussi d'afficher ce résultat en hexadécimal et de choisir le format de ce nombre. A ce point du TP, il est désormais possible d'envoyer tous les caractères de la table ASCII d'origine à un autre ordinateur, de recevoir un message au même format et d'afficher la chaîne de caractères transmise, ainsi que le code hexadécimal de chaque octet reçu.

2) A la découverte de la communication point à point en embarqué.

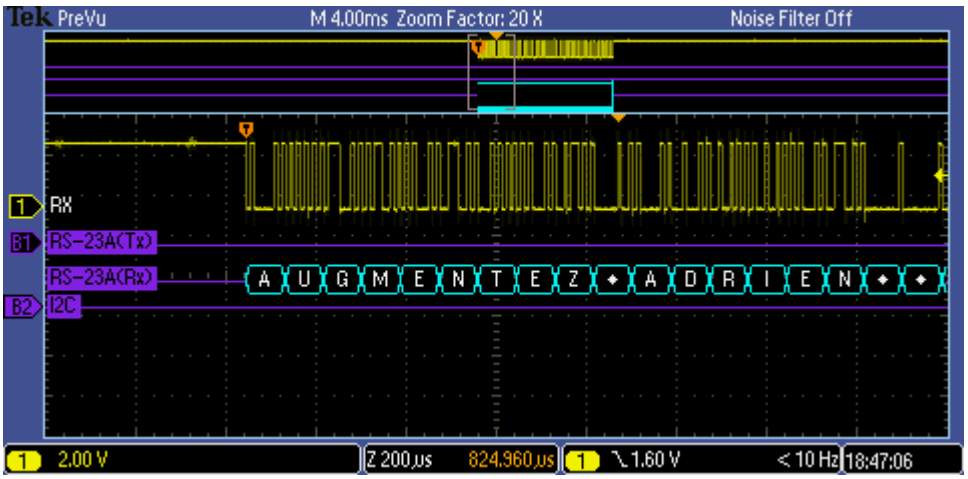
2.1) La liaison série embarquée.

Maintenant que le code en C# permet d'envoyer et de recevoir des informations directement sur l'ordinateur, il faut modifier le programme du robot pour qu'il puisse lui aussi communiquer avec la solution Visual Studio.
Afin de permettre au robot d'échanger des données avec l'ordinateur, il va être nécessaire d'initialiser l'UART présent sur le robot. Un UART est comme son nom l'indique un système de communication série asynchrone admettant un canal de réception et un canal d'émission. Le fait qu'il soit asynchrone signifie que les données transmises devront contenir des octets permettant d'identifier le début et la fin d'une trame. Il faut aussi définir la vitesse à laquelle seront envoyées les informations entre le robot et l'ordinateur. Dans le cas présent, la vitesse sera fixée à 115 200 bauds.

2.2) Echange de données entre le microcontrôleur et le PC.

2.2.1) Emission UART depuis le microcontrôleur.

Pour que le microcontrôleur puisse envoyer les informations qu'il souhaite partager à l'UART, il faut commencer par remapper deux pins qui sont compatibles. Pour cela, les pins 24 et 36 ont été sélectionnés, le premier étant associé à l'entrée Rx et le second à la sortie Tx.
Une fois la liaison physique entre le microcontrôleur et l'UART établie, il faut alors rajouter la fonction qui permet l'envoi des données sur l'UART.
Pour tester le bon fonctionnement du programme d'envoi de données, on ajoute une boucle dans le fichier main permettant l'envoi d'un message sur le port Tx à un intervalle régulier.
Il est alors possible de l'observer à l'oscilloscope pour vérifier que le message est le bon et qu'il n'est pas corrompu. Si l'on se branche sur la borne Tx, on observe le relevé suivant sur l'oscilloscope :



On constate alors que le message affiché correspond parfaitement avec celui indiqué dans la boucle de test. Il est aussi possible de voir que la solution Visual Studio reçoit quelque chose de la part du robot.

2.2.2) Réception.

Pour vérifier que la partie réception du robot fonctionne, il faut aussi utiliser le principe de loopback, mais cette fois-ci de façon logicielle. Dans le cas présent, cela permet de vérifier que ce que le robot renvoie et bien ce qu'il a reçu.
Il faut alors envoyer un message depuis l'interface en C# et attendre de recevoir la réponse du robot sur le port Tx et dans la boite de réception. Il est aussi possible de regarder l'envoi et la réception de la trame sur l'oscilloscope.

2.3) Liaison série avec FIFO intégré.

2.3.1) Le buffer circulaire de l'UART en émission.

Le système d'envoi qui a été mis en place jusqu'ici permet de faire circuler des informations entre le robot et l'interface sur ordinateur. Cependant, sa conception repose sur le principe d'interruption ce qui signifie qu'elle stoppe les autres processus du programme du début à la fin de l'envoi. Or, il n'est pas acceptable de perdre le contrôle du robot lorsque l'on souhaite échanger avec l'ordinateur. Il faut donc revoir la façon dont fonctionne l'envoi d'un message à partir du robot. Pour ce faire, nous allons encore utiliser un buffer de type FIFO, cette fois-ci, le buffer sera dit de type circulaire. Ce type de buffer a pour principale caractéristique que l'endroit où l'information est écrite n'est pas nécessairement le même que celui où l'on vient en lire une. De plus, ce genre de buffer possède une taille prédéfinie et fixe, ce qui permet donc d'optimiser la taille qu'il prend dans la mémoire du système. Ce genre de buffer est donc parfait pour de l'embarqué comme dans le cas du robot actuellement étudié.

L'usage d'un tel type de buffer permet d'envoyer des messages depuis le robot sans interrompre les autres processus en cours. Pour ce faire, il suffit d'entreposer les caractères devant être envoyés dans le buffer et de procéder à l'envoi une fois que le robot aura le temps de le faire. Du fait des caractéristiques du buffer circulaire, il est donc possible de savoir s'il reste des caractères n'ayant pas encore été traités dans la mémoire du robot et de s'en occuper.

Pour mettre en place ce buffer à la place de la méthode d'émission précédente, il faut d'abord modifier le fichier d'initialisation de l'UART pour activer les interruptions sur le Tx.
Le programme permettant de mettre en place le buffer circulaire se base sur les indices de la tête et de la queue de celui-ci. Avec ces deux éléments, il est possible d'ajouter des données dans le buffer et de venir les récupérer après, mais aussi de connaître la place qu'il reste et la taille prise par la donnée en question.

Une fois que tout le programme est mis en place, il est possible de faire le même test que pour la précédente fonction d'envoi et de constater que cette fois-ci le processus ne bloque pas les autres actions que doit réaliser le robot.

2.3.2) Le buffer circulaire de l'UART en réception.

Comme pour la partie précédente, si l'on souhaite que le robot puisse traiter les informations qu'il reçoit sans entraîner l'arrêt des autres tâches, il sera nécessaire d'implémenter un buffer circulaire. Cette fois-ci le buffer permet de stocker les données qui sont reçues sur la liaison série afin qu'elles puissent être prises en compte lorsque le robot pourra le faire.

La mise en place du second buffer ressemble à quelques détails près à celle du buffer d'émission. Il n'est donc pas nécessaire de s'étendre dessus.

Pour tester le bon fonctionnement du second buffer, la fonction qui permettait d'émettre un message depuis le robot est modifiée afin de récupérer les caractères que le robot reçoit et de les renvoyer sur la liaison Tx. Il faut donc ensuite envoyer une trame depuis l'interface Visual Studio et attendre la réponse du robot sur cette même interface ou grâce à l'oscilloscope.

3) A la découverte de la supervision d'un système embarqué.

3.1) Implémentation en C# d'un protocole de communication avec messages.

Le système de messagerie actuellement en place sur l'interface et le robot permet de communiquer tout ce que l'on souhaite. Cependant, cette façon de faire ne garantit pas que les trames ne puissent pas être corrompues. Or, si une trame corrompue est reçue, elle peut alors perturber le fonctionnement du robot ou de l'interface. Il faut donc mettre en place un protocole qui permet de savoir si les informations reçues sont bel et bien celles qui ont été envoyées. Pour ce faire, il est possible de formater les trames afin qu'elles incluent dorénavant des octets dédiés à la vérification de son intégrité.
Il faut d'abord être capable de savoir quand commence une trame. L'ajout d'un octet dit « Start Of Frame » permet de répondre à cette première demande. Le SOF est un octet qui aura toujours la même valeur peu importe la trame qui est émise. Ici, le SOF prend la valeur 0xFE.
Donc lorsque le robot ou l'interface recevra un octet ayant la valeur 0xFE, il saura qu'il s'agit du début d'une trame.
Ensuite, afin que les trames puissent être différenciées en fonction de ce qu'elles sont censées faire, il faut ajouter une indication quant au type de commande. Dans le cas présent, cette information est codée sur 2 octets. Chacune des valeurs de commandes sera alors associée à un type d'action.
Puis, pour s'assurer que le programme puisse traiter correctement les données, il faut indiquer le nombre d'octets correspondants à ces données. Cette information est elle aussi codée sur 2 octets.
Il faut ensuite transmettre les données dont la longueur est inscrite sur les 2 octets précédents. On parle ici de payload.
Enfin, il faut ajouter un octet permettant de contrôler que l'ensemble des octets précédents n'aient pas été corrompus lors du transfert. Cet octet s'appelle le « checksum ». Le principe de ce checksum est le suivant : une fois que toutes les données ont été écrites dans la trame, le checksum effectue une opération de Ou Exclusif bit à bit sur tous les octets et renvoie une valeur qui sera stockée à la fin de la trame. Puis cette valeur sera recalculée par le récepteur et comparée. Si le checksum calculé est égal à celui qui est transmis cela signifie que la trame n'est pas corrompue et qu'elle peut être utilisée. Sinon, cela signifie qu'il y a eu une erreur lors de l'échange de données et qu'il ne faut pas prendre en compte cette trame et le détruire.

3.1.1) Encodage des messages.

Il faut donc écrire une fonction permettant de mettre en place tous les éléments dans le bon ordre et de réaliser les calculs. Le principe est simple, on commence par définir une fonction qui permet de calculer le checksum et que l'on appellera au moment opportun.
Puis, la fonction d'encodage est une simple concaténation des éléments au sein d'une seule trame. On commence par rajouter le SOF en début de trame, puis la commande et la longueur du payload suivie par le payload lui-même, et enfin le checksum.
Cette fonction réalise aussi l'envoi de la trame sur le port série.
Une fois que la fonction est prête, il est possible de vérifier que l'encodage est réussi en le visualisant sur l'oscilloscope. On observe alors que chacun des composants de la trame se trouve au bon endroit et qu'il a la bonne valeur. Nous sommes donc capables d'émettre une trame qui respecte un protocole de communication.

3.1.2) Décodage des messages.

Il faut maintenant être en mesure de lire les messages envoyés grâce à ce protocole. Pour ce faire, il faut ajouter une fonction qui vient faire l'inverse de la fonction précédente. C'est-à-dire que la trame reçue sera décomposée en plusieurs éléments et à chacun de ces éléments associés sera associé sa fonction dans la trame.
Il faut donc commencer par retirer l'octet de SOF, il suffit de récupérer le premier octet qui est reçu et de vérifier qu'il vaut bien 0xFE.
Une fois le SOF reçu, il est possible de récupérer la commande. Pour ce faire, il faut stocker le premier octet reçu dans une variable et penser à décaler celui-ci de 8 bits pour qu'il corresponde bien à l'octet de poids fort. Il faut ensuite attendre l'arrivée du second octet et le rajouter dans le tableau à la place de l'octet de poids faible. La valeur qui résulte de ces deux octets sera utilisée plus tard. Après avoir enregistré le type de commande, il faut récupérer la longueur de payload. Comme pour la commande, il faut créer une variable dans laquelle on met le premier octet de longueur reçu à la place de l'octet de poids fort et attendre que le second arrive pour le mettre à la place de l'octet de poids faible. Une fois ceci fait, il faut alors lire la valeur qui permettra de traiter la réception du payload.
Une fois la longueur du payload connue, il faut stocker le nombre d'octets indiqués dans un tableau d'octet qui fera la même taille que le nombre d'octets contenus dans le payload. Ils sont tous stockés à la suite dans ce tableau.
Lorsque tous les octets du payload ont été stockés dans le tableau d'octets, il est possible de stocker le cheksum reçu dans une variable afin qu'il puisse être comparé après.
Une fois que tous les éléments sont séparés, il est possible de passer au calcul du checksum, et pour ce faire, on répète l'opération de Ou Exclusif bit à bit sur l'ensemble de la trame à l'exception du cheksum. On compare ensuite celui-ci avec le checksum qui était compris en bout de trame. S'ils sont identiques alors la trame est conservée et on vient lire les informations du payload. Sinon la trame est considérée comme corrompues et est détruite.

Nous sommes donc maintenant en mesure d'émettre et de recevoir des messages encodés selon un protocole de communication.

3.1.3) Pilotage et supervision du robot.

Maintenant que les messages respectent un protocole de communication, il est possible de définir une liste d'action permettant de contrôler le robot depuis l'interface Visual Studio.

Il faut d'abord commencer par associer un numéro de commande à chacune des différentes fonctions qui permettent la supervision du robot.

La partie commande de la trame permet d'identifier le type de trame reçu, et ce à quoi correspondent les données du payload. Dans notre cas, nous allons commencer par définir quatre possibilités pour ce numéro de commande.
Si la commande vaut 0x0080, cela signifie qu'il s'agit d'une transmission de textes entre le robot et l'interface. Il n'y a donc pas de traitement supplémentaire à faire sur les données du payload avant de les afficher.
Si la commande vaut 0x0020, cela signifie que l'on souhaite s'occuper des LED. Le payload est alors composé de 2 octets. Le premier permet de savoir quelle LED est concernée, et le second permet de connaître l'état de cette dernière.
Si la commande vaut 0x0030, cela signifie qu'il s'agit des distances des télémètres Infrarouges. Le payload est alors composé de trois octets. Les octets correspondent respectivement aux distances en cm du télémètre gauche, du télémètre central et enfin du télémètre droit. Les valeurs des télémètres extrêmes ne sont pas transmises dans le payload.
Enfin, si la commande vaut 0x0040, cela signifie que l'on souhaite connaître la vitesse des moteurs. Le payload est alors composé de trois octets. Le premier octet indique le pourcentage de vitesse du moteur gauche et le second celui du moteur droit.

Avec toutes ces fonctions, il est possible de connaître toutes les informations importantes permettant de superviser le robot depuis l'interface utilisateur.

Il faut alors modifier l'interface graphique pour qu'elle puisse afficher ces valeurs et permettre d'en modifier certaines.

3.2) Implémentation en électronique embarquée.

Maintenant que l'interface Visual Studio peut envoyer et recevoir des messages avec un protocole de communication, il est nécessaire de mettre à jour le programme du robot pour qu'il puisse en faire autant.

3.2.1) Supervision.

Il faut donc commencer par ajouter la couche de protocole à l'UART. Pour ce faire, il suffit de rajouter les fonctions d'encodage et de décodage des trames. Les fonctions sont similaires à celles produites en C#, mais nécessitent certains ajustements afin de pouvoir fonctionner en C.
Une fois ces fonctions réalisées, il est possible d'essayer de communiquer avec l'interface. On simule alors l'envoi d'un texte depuis le robot et on observe que le texte apparaît bien sur la boite de réception.

Il faut ensuite vérifier que l'on peut lire les valeurs de distances des télémètres Infrarouges. Pour ce faire, il faut retirer la simulation de message précédemment ajoutée et la remplacer par l'envoi d'une trame à la fin de la fonction de conversion des données des télémètres Infrarouges. On observe alors le message suivant sur la boite de réception « Cpt Gauche : 50 cm ; Cpt Centre : 35 cm ; Cpt Droit : 47 cm. ».

Si l'on souhaite pouvoir mieux superviser le robot, l'ajout d'une fonction permettant de savoir dans quel état il se trouve est intéressant. Cette commande aura la valeur 0x0050. Le payload sera alors composé de cinq octets, un permettant de connaître le numéro de l'étape, et quatre pour savoir le temps écoulé depuis la mise en marche du robot.

Il faut donc ajouter cette même fonction en C# pour que l'on puisse l'afficher sur l'interface utilisateur.

3.2.2) Pilotage.

Une fois que la supervision du robot est possible, il faut implémenter les fonctions permettant d'agir directement sur le robot pour pouvoir le piloter.

Il faut d'abord commencer par permettre au robot de pouvoir décrypter les informations envoyées depuis l'interface Visual Studio. Pour ce faire, on s'inspire du programme en C# et on l'adapte en C afin de pouvoir le rajouter sur le robot.
Après avoir réalisé cette fonction de décodage, il faut modifier la structure de base du programme du robot afin de pouvoir prendre le contrôle via l'interface lorsqu'on le souhaite.
Il faut donc ajouter une variable qui permet de décider quand le robot passe du mode automatique au mode manuel. Cette variable est définie de façon que le robot commence toujours en mode automatique. Cette variable doit aussi être présente sur l'interface utilisateur.

Via l'ajout de nouvelles valeurs de commande, il est dorénavant possible de choisir si le robot fonctionne en automatique ou en manuel, et dans le cas du mode manuel, il est possible de choisir dans quel état se trouve le robot. Ces nouvelles valeurs sont 0x0052 pour le choix du mode, et 0x0051 pour le choix de l'état. Elles sont ajoutées dans le fichier UART_Protocol du robot et dans le switch case du programme en C#.

Il est donc maintenant possible de prendre le contrôle du robot en envoyant une trame de type 0x0052 pour une valeur du payload de 0, et de le remettre en mode auto par l'envoi d'un même type de trame mais avec un payload valant cette fois-ci 1. Puis, une fois en mode manuel, l'envoi d'une trame de type 0x0051 permet de forcer l'état du robot en fonction de l'état mis dans le payload.

À l'issue de ces séances de travaux sur le robot et le logiciel Visual Studio, nous sommes à présent en mesure de superviser et de contrôler le robot depuis l'interface graphique que nous avons mis au point. Nous avons compris le fonctionnement de plusieurs outils utilisés en programmation comme les buffers et la mise en place de protocoles de communication et les avons appliqués sur le projet.

4) Ajouts potentiels.

Maintenant que nous sommes capables d'utiliser l'interface conçue sur Visual Studio pour faire faire des actions au robot, la prochaine étape serait de pouvoir contrôler le robot plus en profondeur. Cela pourrait se faire par le contrôle direct des commandes de vitesses des moteurs via le clavier de l'ordinateur avec des contrôles type jeux vidéo afin de pouvoir gérer l'accélération et les freinages en direct. En ajoutant deux autres touches pour pouvoir gérer les virages, il sera alors possible de réellement piloter le robot depuis un ordinateur plutôt que de jouer avec les états déjà présents pour tenter de lui donner la direction que l'on souhaite qu'il suive.

Dès lors que le contrôle par clavier est maîtrisé, pourquoi ne pas rajouter une étape encore au-dessus où l'on pourrait alors relier une manette ou un contrôleur de même type afin d'avoir une meilleure prise en main des déplacements du robot. L'ajout de joysticks permettrait encore un gain en contrôle du robot aussi bien pour les accélérations et les freinages, mais aussi pour le contrôle de la trajectoire.

Toujours dans l'optique d'améliorer le pilotage du robot, il faudrait alors revoir l'interface en elle-même. L'ajout de nouvelles données telles que celle de la vitesse réelle calculée à partir des capteurs à incrémentation mis en place sur les roues du robot, ou encore un système d'affichage similaire aux détecteurs de recul d'une voiture permettraient de faciliter les manœuvres avec le robot sans pour autant avoir besoin de le garder en visuel. Il faudrait évidemment pouvoir contrôler le robot sans y être directement branché.