INTERFACE GRAPHIQUE DE CONTRÔLE POUR LE ROBOT LEENBY

Aujourd'hui, la seule application permettant de contrôler le robot Leenby n'est utilisable que sous Windows. Le but de cette interface, est de fournir un moyen de contrôler le robot Leenby à distance avec un ordinateur sous Linux, en utilisant ROS.

Pour cela, un package hybride ROS-Qt est utilisé. L'interface est codée en C++ avec l'utilisation des widgets Qt, ainsi que certains widgets personnalisés comme le joystick. En parallèle de cela, des informations de commande sont publiées sur des topics ROS.

Dans cette documentation, nous expliquerons le fonctionnement général du point de vue de l'utilisateur, puis nous inspecterons morceau par morceau le code Qt permettant l'affichage, et enfin le code permettant le traitement et la communication des informations.

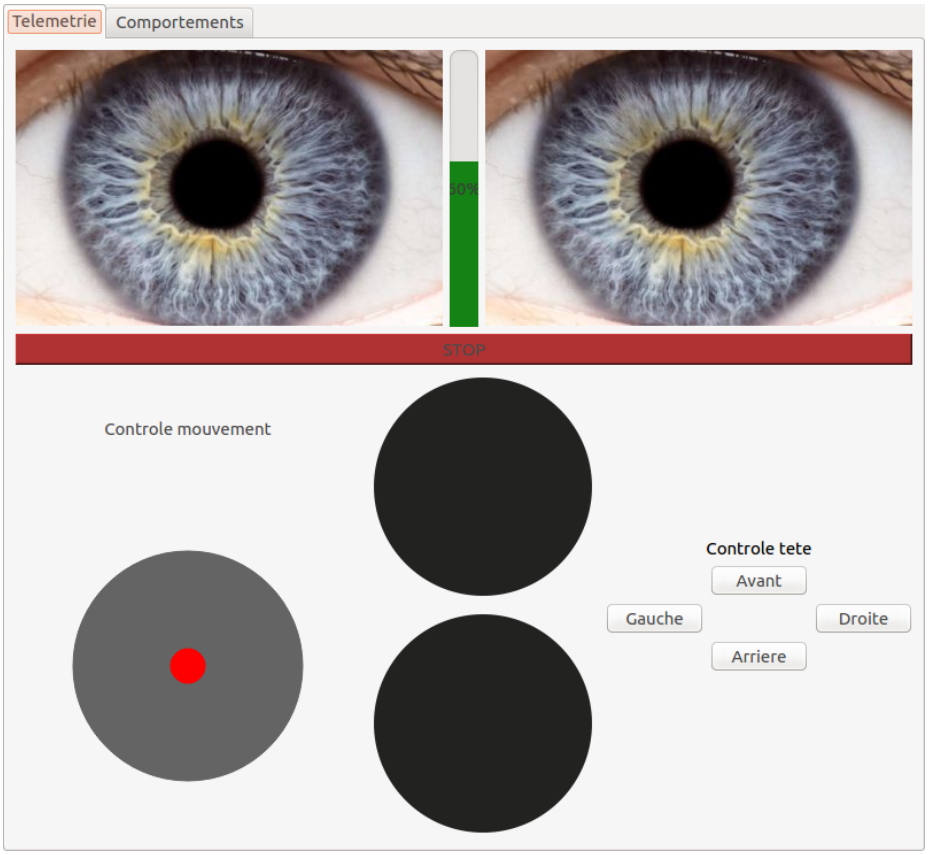

Illustration 1: L'interface

Présentation générale

Commençons d’abord par lister les fonctionnalités prévues pour cette interface :

- Affichage des retours des caméras
- Affichage du niveau de batterie restant (Pas fonctionnel)
- Bouton d’arrêt d’urgence
- Joystick
- Affichage des retours des lidars
- Contrôle des mouvements de la tête
- Intégration de Rviz pour visualiser les données
- Envoie de message pour que le robot parle

Description des fonctionnalités :

Les deux images d’yeux en haut à gauche et en haut à droite de l’illustration 1 sont les emplacements destinés à accueillir les retours des caméras. Les images sont récupérées sur un topic ROS. Tant qu’aucune image n’est reçue, les images d’yeux seront présentes. Une fois la première image reçue, l’image disparaît et le retour caméra s’affiche. Les deux images représentant les deux caméras de la Leenby, il est nécessaire d’avoir deux topics ROS.

L’affichage du niveau de batterie restant se fait via la barre verte en haut au centre de l’interface. Cette fonctionnalité n’est pas encore mise en place. Ce qui est prévu, est simplement de récupérer la valeur de batterie restante qui serait publiée sur un topic ROS, puis de mettre à jour cette valeur dans l’interface.

Le bouton d’arrêt d’urgence : ce bouton rouge a pour but de stopper la Leenby en cas de danger. Pour cela, un message de commande est envoyé pour stopper les moteurs, puis tous les widgets permettant de contrôler la Leenby sont désactivés.

Le joystick est un widget, et peut donc être dupliqué. Nous pourrions par exemple utiliser un joystick à la place des boutons permettant le contrôle des mouvements de la tête. Ce joystick permet d’obtenir deux valeurs : une pour la vitesse linéaire, et une pour la vitesse de rotation. Ces deux informations subissent ensuite un léger traitement, avant d’être publiées.

La bibliothèque librviz est utilisée pour afficher le retour des lidars. Dans le premier onglet, il s’agit seulement de l’affichage. La bibliothèque rviz permet d’obtenir une visualisation du retour. Dans le deuxième onglet, une fenêtre beaucoup plus complète et modifiable permet de choisir ce que l’on veut observer et de changer certains paramètres, comme dans Rviz.

Les quatre boutons présent sur la droite de l'interface permettrons dans le futur de contrôler les mouvements de la tête de la Leenby, toujours en publiant des informations de contrôle sur un topic ROS.

Finalement, le troisième onglet permet de publier un message pour faire parler la Leenby.

Pour accéder à certaines fonctionnalités importante, des raccourcis clavier ont été mis en place :

Quitter	Ctrl + q
Arrêt d'urgence	Space
Réactiver les contrôles	Ctrl + Alt + Space
Activer/Désactiver la manette	Ctrl + j
Tête avant	z
Tête arrière	s
Tête gauche	q
Tête droite	d

Tableau 1: Raccourcis clavier

Comportements du joystick :

L'objectif a été d'avoir un joystick sous forme de widget, avec un comportement aussi réaliste que possible. Les principales règles pour ce mimétisme ont été les suivantes :

1) Devoir attraper la partie mobile du joystick pour pouvoir la déplacer.

2) La partie mobile du joystick ne doit pas pouvoir sortir d'un cercle défini

3) Si le curseur de l'utilisateur sort du cercle défini, la partie mobile doit être positionnée à sa distance maximale, et toujours suivre le curseur.

Ensuite, l'information envoyée par le joystick est déterminée en fonction de la position de la partie mobile.

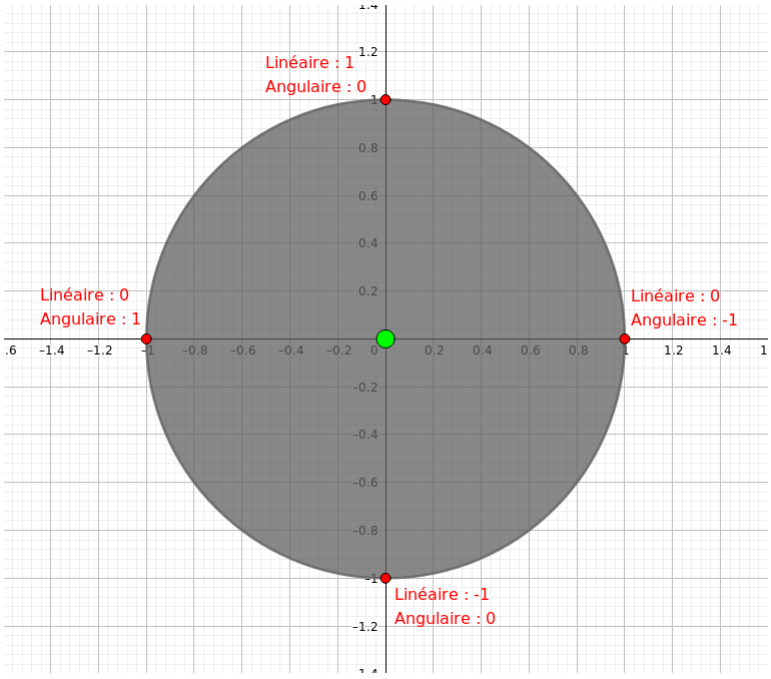

Illustration 2: Graphique de comportement du joystick

Comme nous pouvons le voir sur l'illustration 2 ci-dessus, la commande linéaire est déterminée en fonction de la position sur l'axe Y de la partie mobile (sur cette illustration en vert), et la commande angulaire est déterminée en fonction de la position sur l'axe X de la partie mobile.

Pour plus d'ergonomie, ce joystick peut également être contrôlé par un manette physique. Pour cela il faut brancher la manette et lancer un nœud qui publie l'état de cette dernière. L'interface est prévue pour gérer une manette de Xbox 360 filaire.

L'affichage (C++ et Qt)

La répartition des éléments graphiques dans les containers est assez simple, elle se fait en trois onglets :

Dans le premier, un QBoxLayout englobe la totalité de la fenêtre. Ce layout est ensuite divisé en trois QHBoxLayout, qui accueillent respectivement, les retours caméras ainsi que l'afficheur du niveau de batterie, le bouton d'arrêt d'urgence, le joystick ainsi que les retours des lidars et les boutons de commande de la tête.

Une représentation de ces containers pourrait être la suivante :

<table>
<tr><td>Caméra gauche</td><td>Batterie</td><td>Caméra droite</td></tr>
<tr><td colspan="3">Bouton d'arrêt d'urgence</td></tr>
<tr><td>Texte
Joystick</td><td>Retour lidar tête
Retour lidar base</td><td>Texte
<table><tr><td></td><td>Bouton avance</td><td></td></tr><tr><td>Bouton gauche</td><td></td><td>Bouton droite</td></tr><tr><td></td><td>Bouton recul</td><td></td></tr></table></td></tr>
</table>

Tableau 2: Répartition des widgets dans les containers

Dans le second onglet, le widget Rviz et seul et finalement le troisième onglet contient une zone de texte et un bouton. La zone de texte permet d'écrire le message que l'on veut faire dire au robot et le bouton permet de publier ce message.

Cette disposition est celle qui, au moment de la rédaction de la documentation, fonctionne correctement. En fonction de l'évolution de l'implantation des nouvelles fonctionnalités, cette disposition pourra être modifiée.

Les éléments :

Les retours des caméras sont des QLabel. Au départ ils sont des images, puis quand une image des caméras est reçue, le QLabel accueil le retour de la caméra.

Pour ce qui est de l'affichage du niveau de batterie restant, nous avons utilisé une QProgressBar positionnée verticalement, de couleur verte, et pouvant afficher des valeurs comprises entre 0 et 100. Ces valeurs sont exprimée en pourcentage de batterie restante.

Le bouton d'arrêt d'urgence est simplement un QPushButton de couleur rouge. Du fait que ce bouton est placé seul dans un QHBoxLayout, il s'étendra jusqu'à le remplir. Cela permet d'avoir un bouton large et visible, facilement accessible en cas de besoin.

Le joystick est un widget personnalisé du type JoystickWidget qui hérite de la classe QWidget. Il s'agit de deux cercles, le gris correspond à la surface d'utilisation du joystick, et le rouge est l'élément mobile. Ces cercle sont redessinés à chaque nouvel événement lancé par la souris.

Le retour des lidars est géré par un widget de type LidarView. Ce type hérite de RvizWidget, qui lui est créer par la librairie Rviz est permet d'intégrer une fenêtre de visualisation 3D dans l'interface.

Finalement, un QGridLayout contient le texte et les boutons qui contrôlent la tête de la Leenby.

Ensuite dans le deuxième onglet, il s'agit d'un simple RvizWidget. Et dans le troisième onglet, on peut écrire quand un QTextEdit et envoyer l'information avec un bouton simple.

Le traitement des informations (C++ et ROS)

Le joystick :

Une documentation complète propre au joystick à été rédigée. Pour plus d'information, s'y référer.

Le RvizWidget :

Une petite documentation est disponible pour ce widget.

La transmission des informations :

Dans le fichier qnode.cpp :

Utilité	Type	Type message	Topic	Callback	Signal emitted
Message parole	Publisher	std_msgs::String	/messageToSay	x	x
Message commande	Publisher	geometry_msgs:: Twist	/cmd_vel	x	x
Image caméra gauche	Subscriber	sensor_msgs::Image	/camera/image	imageCallbackGauche	Qnode::hasReceivedImage
Image caméra droite	Subscriber	sensor_msgs::Image	/camera/image	imageCallbackDroite	Qnode::hasReceivedImage
Commande manette	Subscriber	sensor_msgs::Joy	/joy	manetteCallback	Qnode::manetteMoved(double,double)