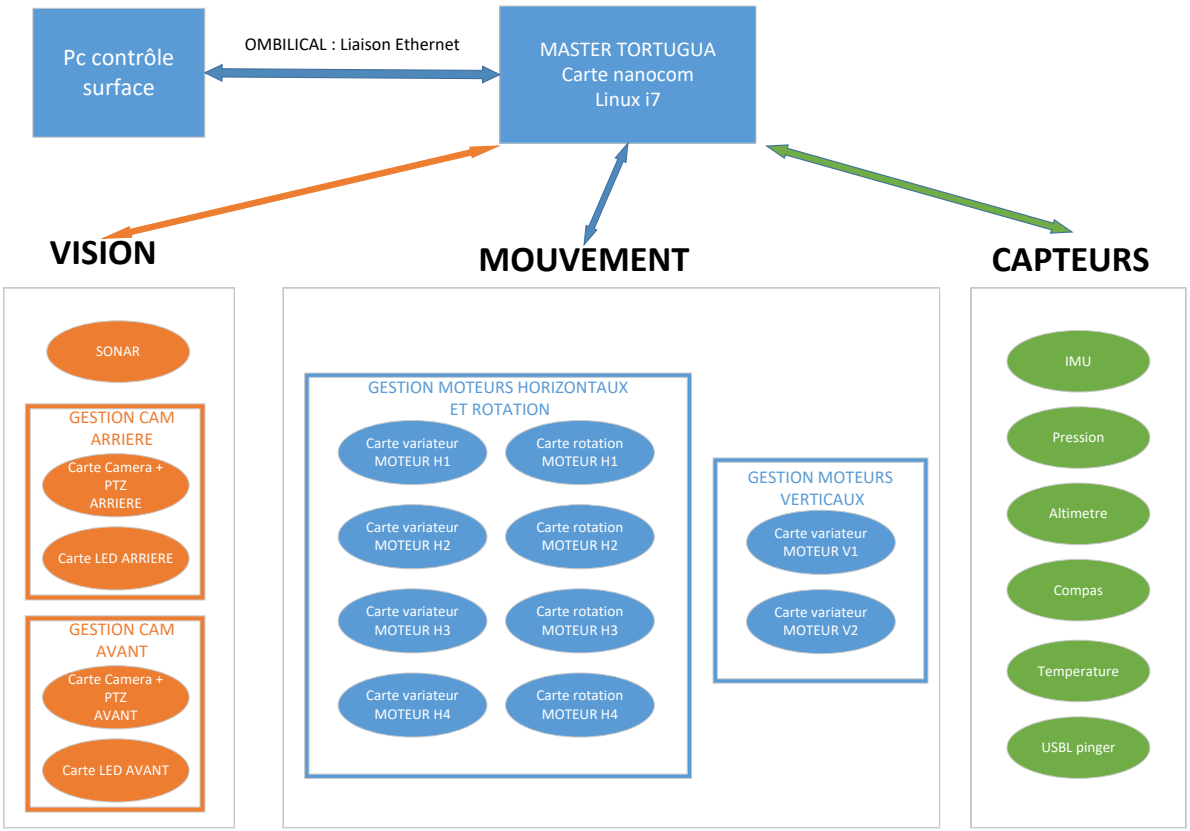
Table des matières

1. Description Synoptique informatique Tortugua



2. Le cœur de Tortugua

La carte Nanocom i7 constitue le cœur de tortugua qui sera en charge du contrôle de l'engin et de la communication interne des différentes cartes électroniques qui le compose mais également de la gestion et de la communication avec le pc de surface.

Système de la carte principale Nanocom :

- Système d'exploitation **LINUX UBUNTU 16.04**
- **ROS** version **KINETIC**
- **QT** version 5.7 c++
- Listes (à compléter) des librairies installées :
  - Opencv 3.1
  - BOOST 1.35
  - SocketCan
  - …

Le système Tortugua sera composé de plusieurs modules « ROS » indépendants ainsi que d'un superviseur général (Le master) qui sera en charge de coordonner l'ensemble des actions et la communication avec le pc de surface.

Liste des non – exhaustives des fonctionnalités nécessaires :

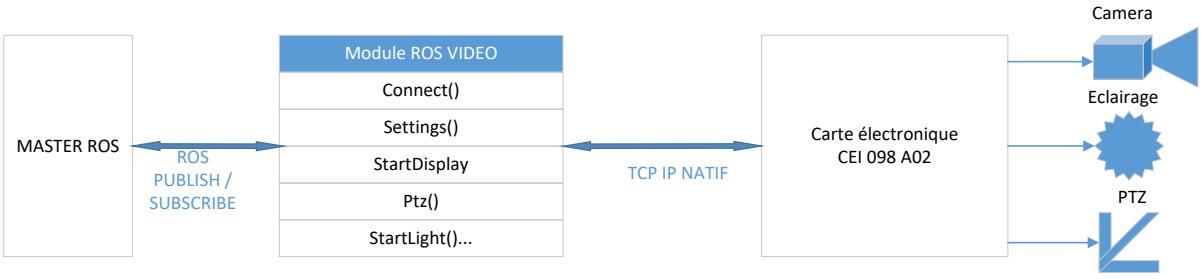
- Contrôle des moteurs de l'engin.
- Gestion du mode autonome de navigation.
- Activation, configuration et retour des capteurs vidéo (caméras + sonar).
- Commande des PTZ Avant/Arrière.
- Gestion des éclairages.
- Retour des capteurs d'environnements (température, pression, attitude …).
- Enregistrement des paramètres internes de l'engin.
- Watchdog + gestion d'erreurs (Perte de com, pb électroniques …).

3. Les modules de vision

Le développement de la vision pour tortugua se décompose principalement en deux parties distinctes : la gestion Vidéo et la gestion Sonar.

a. La vidéo

Le développement est constitué d'un module indépendant ROS de gestion de la vidéo et de l'éclairage. Ce module sera en charge de la connexion, de la communication et du retour vidéo mais également de la gestion du ptz associé à une caméra.

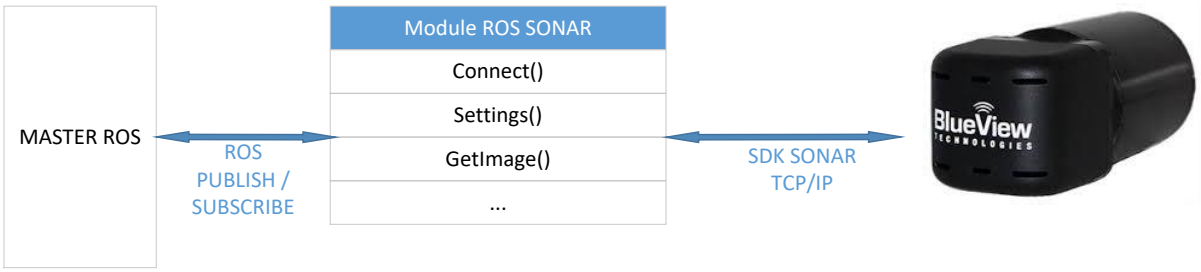


On pourra s'abonner (subscribe) à la donnée « Image_video » pour obtenir la vidéo que ce soit en surface ou sur l'engin. Et on publiera (publish) des commandes pour le PTZ et l'éclairage à partir d'un module ROS interne ou externe à l'engin.

Publish()	Std_msgs :Bool start/stop	Std_msgs: int Pan /Tilt	Std_msgs: int Val_light	
Subscribe()	Sensor_Msgs:Image Image_camera	Std_msgs:int Retour_intensite		

b. Le sonar

Le développement est constitué d'un module indépendant ROS de gestion de la connexion et de l'acquisition sonar.



On pourra s'abonner à la donnée « Image_sonar » pour obtenir l'image que ce soit en surface ou sur l'engin. Et on publiera des commandes de Start/Stop et de configuration à partir d'un module ROS interne ou externe à l'engin.

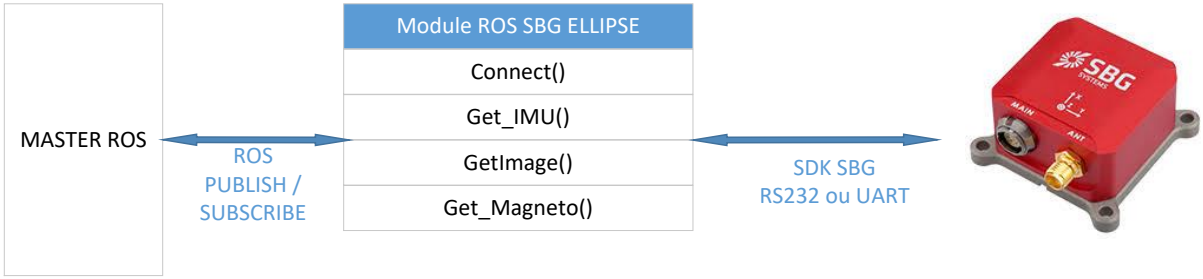
Publish()	Std_msgs :Bool start/stop	Std_msgs: int RangeMin/RangeMax	Std_msgs: int Type_color	
Subscribe()	Sensor_Msgs:Image Image_sonar_raw	Sensor_Msgs:Image Image_sonar_Colorized		

4. Les modules des capteurs

On décrit ici les différents modules ROS qui permettent de communiquer et de récupérer les informations des capteurs « d'environnements » pour les afficher et/ou les enregistrer.

a. Le module IMU + Compas

Un module indépendant ROS intègre le SDK fournit par SBG pour communiquer et récupérer les différentes mesures des capteurs de la centrale. (Un driver « sbg_driver » existe déjà dans les packages ROS).

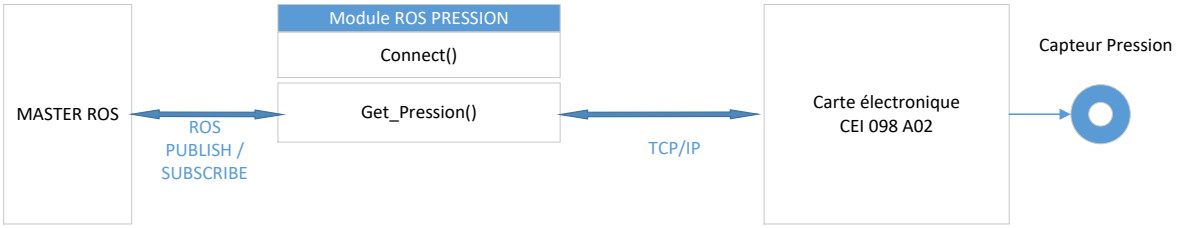


On pourra s'abonner à plusieurs valeurs différentes renvoyées par la centrale.

Publish()	Std_msgs :Bool start/stop		
Subscribe()	Sensor_Msgs:Imu Imu_raw_pub	Geometry_msgs:PoseStamped Ekf_pose_pub	Geometry_msgs :QuaternionStamped Ekf_quat_pub

b. Le module de pression

Ce module permet de lire à un intervalle de temps prédéfini la valeur renvoyé par le capteur de pression.

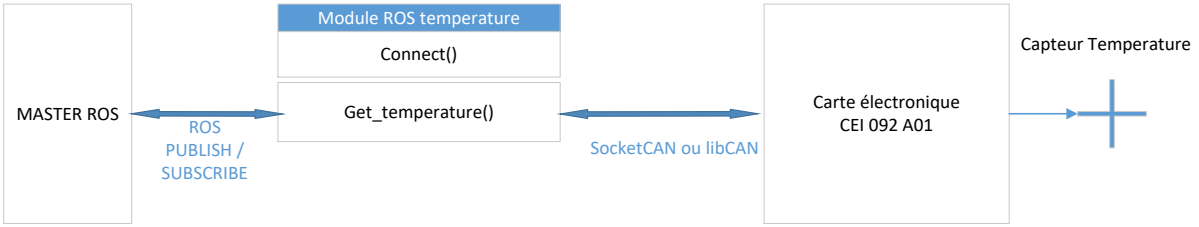


On a donc une unique valeur en retour :

Publish()	Std_msgs :Bool start/stop	
Subscribe()	Sensor_Msgs:FluidPressure (float64) Pression	

c. Le module temperature

Plusieurs capteurs de températures seront présents dans l'engin, nous prenons pour exemple ici le capteur d'une carte moteur. La communication avec le capteur s'effectue sur le bus CAN donc l'implémentation de ce protocole sera soit du SocketCan (directement implémenté dans ROS) ou une librairie générique linux CAN.



On a ici aussi une unique valeur en retour :

Publish()	Std_msgs :Bool start/stop	
Subscribe()	Sensor_Msgs:Temperature (float64) temperature	

d. Le module USBL

A faire……

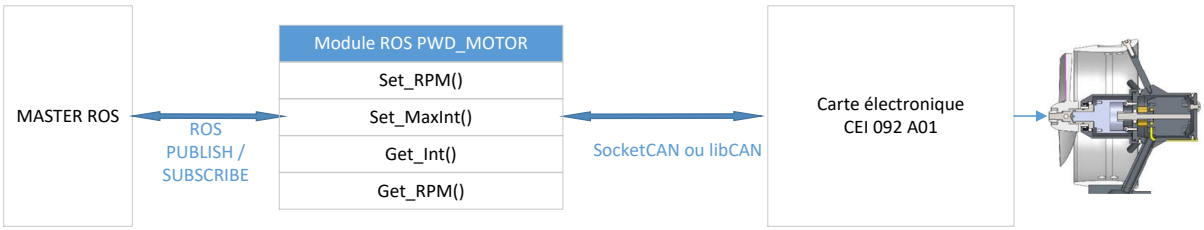
5. Les modules de mouvement

Le développement de la partie « mouvement » de tortugua se divise en plusieurs parties distinctes :

- La communication des ordres aux cartes de gestion des moteurs.
- La navigation en fonction de la configuration moteur.
- La navigation autonome (positionnement dynamique…)

a. Gestion moteurs

Ce module ROS duplicable en fonction du nombre de moteurs permet uniquement de communiquer un ordre de puissance moteur à la carte électronique associée.

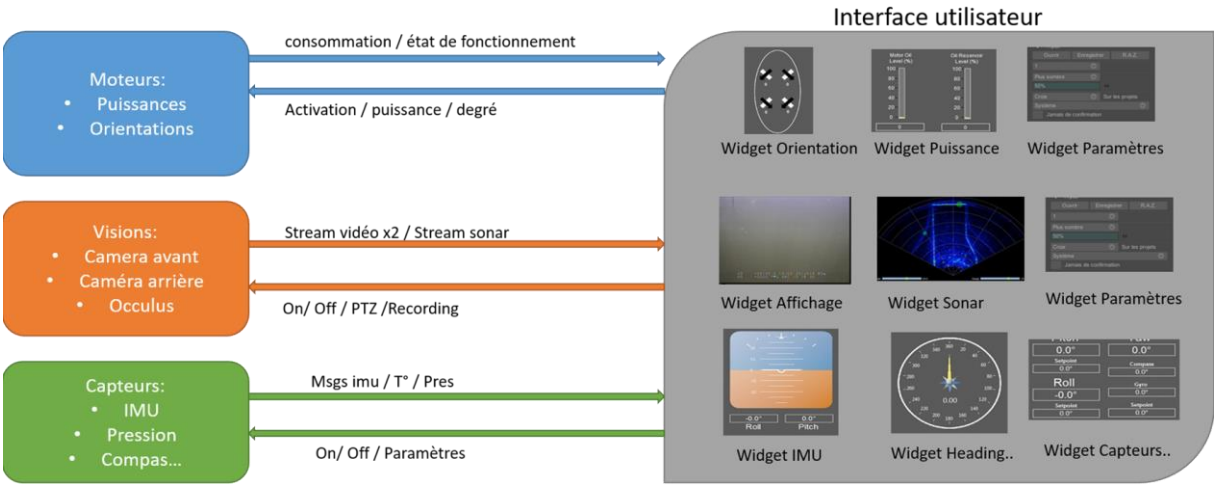


6. L'interface de contrôle

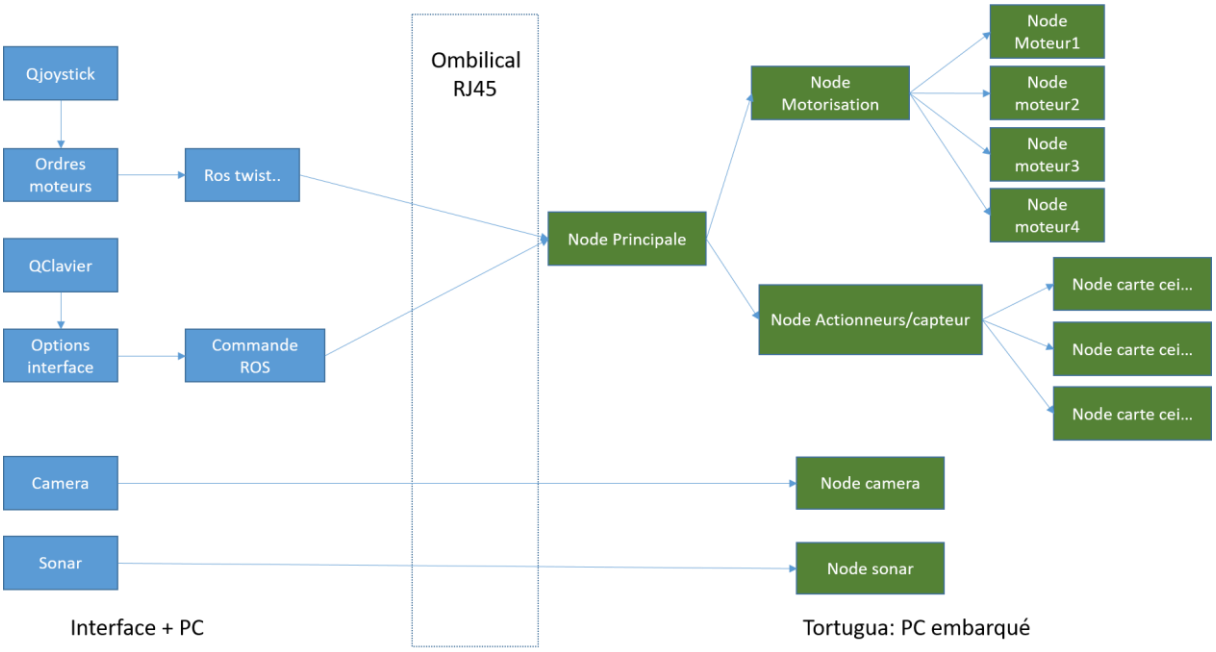
Rappel rapide des fonctionnalités nécessaires à l'utilisateur:

- Pilotage de l'engin
- Visualisation des caméras, sonar.
- Affichage des informations de positions, d'attitude, de profondeur et de vitesse.
- Configuration de l'orientation moteurs.
- Affichage des informations de santé de tortugua
- Activation et utilisation des options et enregistrements

Le système d'exploitation de l'engin tournant sous linux + Ros, l'interface doit être lancé de manière automatique et autonome au démarrage du poste de pilotage (Daemon et affichage fullscreen).



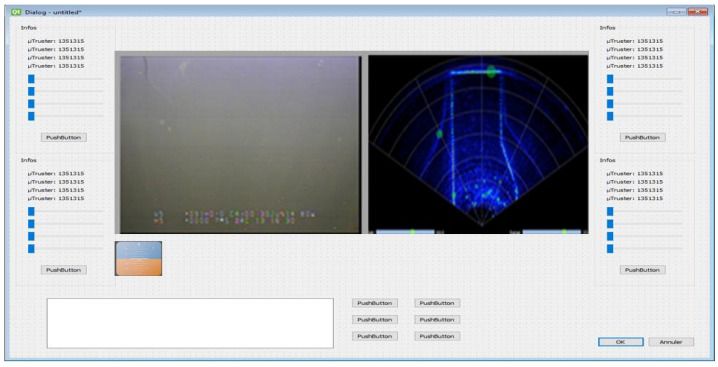
a. Synopsis de l'architecture du système/interface



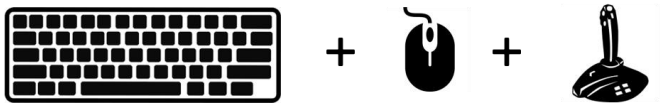
b. Les différentes phases d'interfaces

Cependant lors de la première phase de conception l'interface utilisateur développé correspondra à une interface dite de débogage. Cette interface ne représentera pas le style final mais devra permettre plusieurs fonctionnalités spécifiques au débogage :

Première version de l'interface et des contrôles pour mise au points de l'engin et premiers tests:



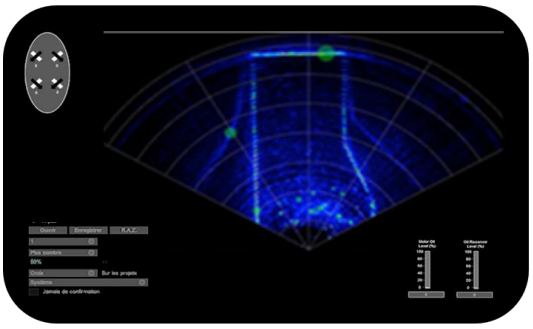
Ecran pc 1



- Affichage et modification en temps réel du maximum de paramètres et capteurs possibles de l'engin.
- Affichage des données réseaux (si possible vitesse et QOS).
- Des fonctions de gestions et d'affichages d'erreurs.
- Des modes manuels d'orientations et de puissances moteurs.
- Utilisation basique du joystick et raccourcis clavier et/ou souris sur l'interface.

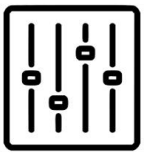
La deuxième phase consistera à mettre au point et à tester l'engin en situation, c'est-à-dire en eau avec courant. Lors de cette phase il apparait nécessaire de développer une interface de contrôle qui permette au pilote de se rapprocher au maximum du contrôle d'un ROV en mission. Pour se faire une maquette d'interface avec joysticks courts, faders et boutons sera développée. Cette maquette représentera dans l'approche fonctionnelle la console de contrôle finale.

- Deuxième version des contrôles pour utilisation en eau et tests vectoriels:



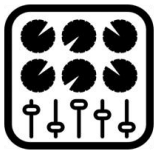
Ecran 1

Ecran 2



Boitier fader USB

+



Boitier bouton USB

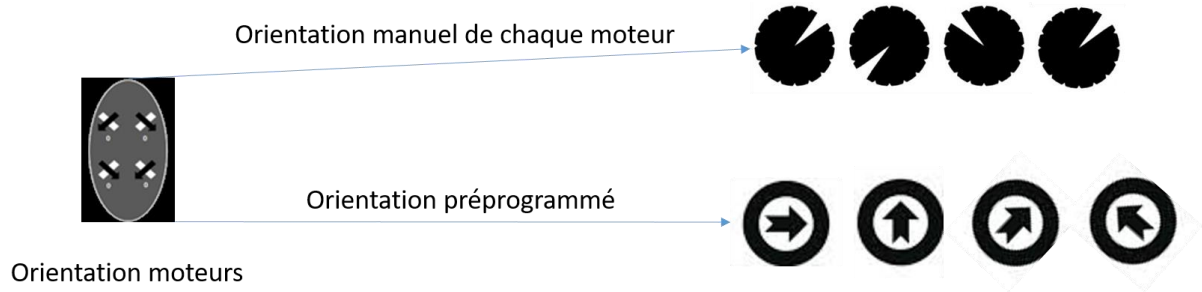
+

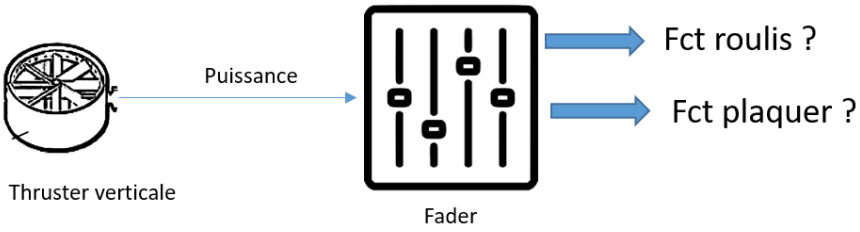
Joystick court

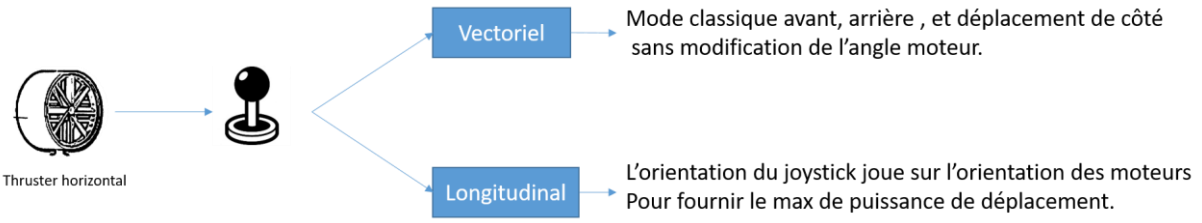
<u>Les specs générales d'interface</u> :

<u>Affichage</u>	**<u>Pan/Tilt</u>**	**<u>Recording</u>**	**<u>Eclairage</u>**
Sélection affichage dynamique et configurable.	Pan/tilt sur joystick.	Bouton de démarrage de l'enregistrement.	Boutons de réglages +/- puissances lumineuses
Bouton physique de switch	Bouton de RAZ.	Affichage de l'état en incrustation	Activation avant/arrière + mode auto en fct de l'affichage
Incrustation paramétrables.	Option position constante ou retour à 0.	bouton de snapshot.	
Affichage des capteurs activables.	Activation dynamique en fonction de l'affichage.	Paramètres d'enregistrement.	
Messages de monitoring et d'erreurs bien visibles.			

<u>La gestion de la motorisation et des modes automatiques</u> :







L'engin possédera également d'autres fonctions automatiques activables depuis l'interface visuelle et physique.

- « **Auto depth** » , « **Auto head** » :
  - Activation automatique ou possibilité d'entrer un ordre.
  - Commande manuelle prioritaire sur l'automatisme.
- « **Lock** »de la commande ou auto fwd :
  - Possibilité de conserver l'ordre donné au moteur sans action sur le joystick.
- Fonction « **Go to** » et « **DP** ».