Projet transversal MEA 4 S7

Dashboard numérique pour voilier

Manuel d'utilisation et d'interfaçage de capteurs

Prêcheur Bonneau Bastin Bordoux
04/02/2020

Table des matières

INTRODUCTION

Ce projet a pour but de créer un Dashboard pour voiler, en d'autres termes un panneau de commandes capable d'afficher différentes données utiles à la navigation d'un voiler comme sa direction ou son orientation.

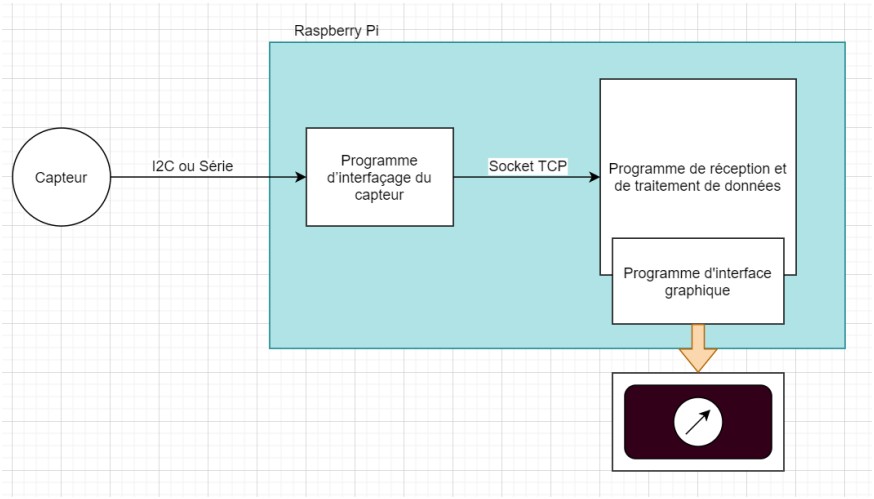
A l'heure actuelle, ce programme ne prend en charge que la fonction boussole et la fonction accéléromètre détectant les inclinaisons latérales. Mais sa conception à été faite de sorte de simplifier au maximum l'ajout d'une nouvelle fonctionnalité. Ce rapport à donc pour but de vous guider dans l'ajout d'un nouveau capteur dans le système.

Dans ce document nous prendrons comme exemple l'interfaçage d'une boussole.

PRESENTATION GENERALE DU SYSTEME

Le système du Dashboard se voulant le plus modulaire possible, celui-ci est composé de plusieurs parties avec chacune une fonctionnalité différente. L'utilisation du principe de programmation orientée objet était donc la meilleure solution. Le but était de créer une interface graphique et nous avons donc fait le choix d'utiliser la librairie Qt proposant de nombreuses fonctionnalités donc un IDE adapté avec la possibilité de modifier son interface graphiquement : QTCreator.

L'architecture générale est décrite comme ceci :



Le système est donc composé de 3 programmes :

1. Un programme d'interface propre au capteur
2. Un programme de réception et de traitement de données
3. Un programme de génération d'interface appelé par 2.

Le premier programme s'occupe de l'interface entre le capteur connecté en I2C ou en série avec la Raspberry. Il est propre au capteur et ne sera abordé que pour sa partie de transmission de données.

Le deuxième est chargé de réceptionner les données du premier programme et de les traiter pour qu'elles soient utilisables pour le dernier programme appelé par celui-ci (C++ avec Qt).

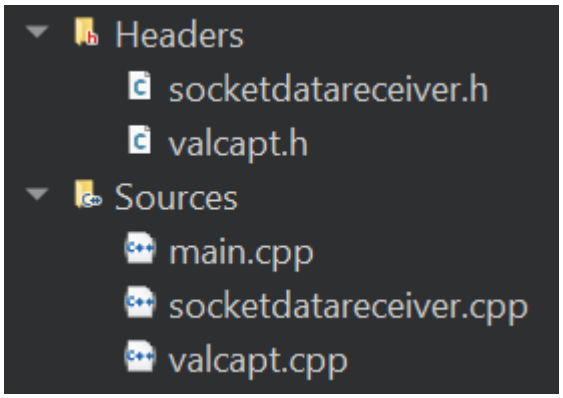
Le troisième est un programme de description graphique propre à la librairie Qt, il récupère les données du deuxième programme et anime les images selon la description apportée dans son code (QML).

PARTIE RECEPTION ET TRAITEMENT DE DONNEES

Dans cette partie, nous traiterons le fonctionnement du programme de réception et le traitement de données et de l'ajout d'un capteur au système.

Fonctionnement

Le programme est un programme C++ faisant appel à la librairie Qt permettant de générer des interfaces graphiques. Le programme orienté objet se structure de cette manière :



1. main.cpp : permet l'appel des différentes classes utiles au fonctionnement du programme et du programme d'interface graphique (VOIR ANNEXE I)

2. socketdatareceiver.cpp/.h : classe créée ayant pour fonction la connexion aux différentes sockets reliées au socket et à la lecture des données dans celles-ci (VOIR ANNEXE II)

3. valcapt.cpp/valcapt.h : classe créée ayant pour fonction de mettre à jour les valeurs des données reçues pour l'animation des cadrans (VOIR ANNEXE III)

Ajout d'un capteur

Pour rajouter un capteur au programme il suffit de suivre les instructions en commentaires dans le code. Les capteur boussole, accéléromètre et GPS sont déjà instanciés. Deux autres capteurs arbitraires sont ajoutés et commentés. Il suffit simplement de décommenter les lignes indiquées dans valcapt.cpp et valcapt.h pour les ajouter.

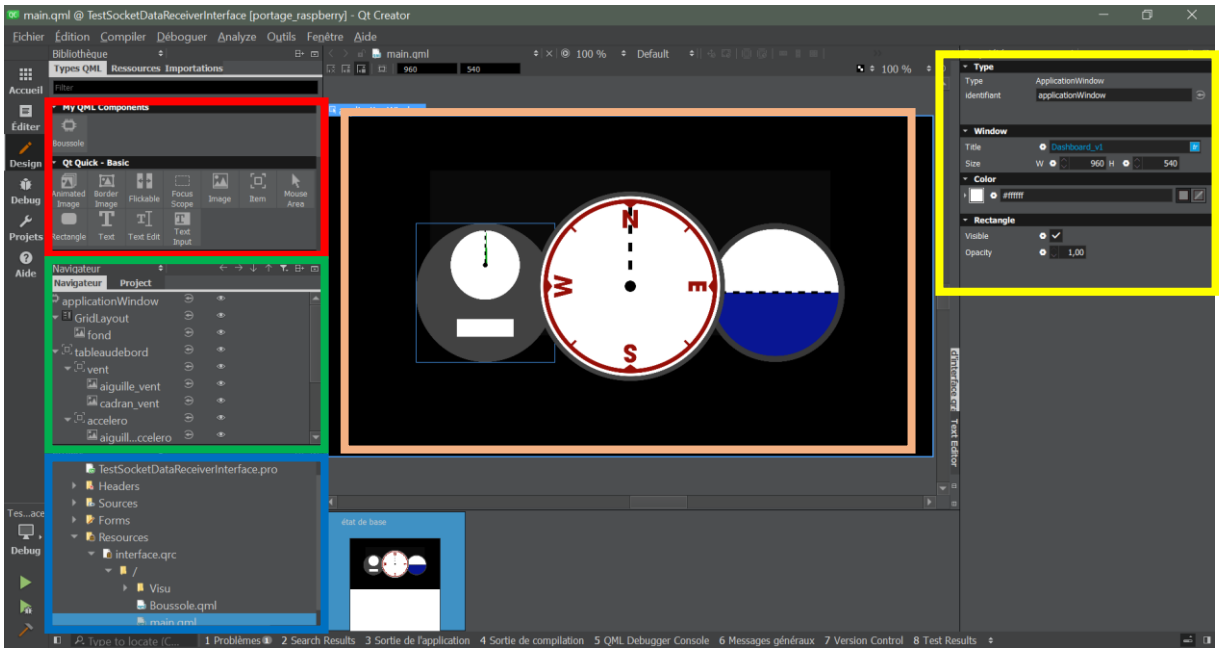
Il est bien sur possible de rajouter encore plus de composants. Il suffit de trouver un port socket TCP libre et de l'indiquer à l'endroit prévu à cet effet.

PARTIE INTERFACE GRAPHIQUE

Dans cette partie nous traiterons du fonctionnement du programme de génération de l'interface graphique et de l'ajout d'un cadran à celle-ci.

Fonctionnement

Ce programme est un programme en QML. C'est un langage de description graphique propre à Qt. Ce langage est directement lié à une interface de développement graphique QTDesigner permettant de décrire nos fenêtres de manière graphique. QTDesigner est inclus dans QTCreator et se présente comme ceci :



Visualisation de la fenêtre

Banque de composants à ajouter à la fenêtre

Arborescence des composant dans la fenêtre

Propriétés du composant sélectionné

Arborescence du projet

Lors de la modification de la fenêtre dans cette interface, le code QML va être généré automatiquement, il n'y a donc rien à faire sur ce code mise à part l'animation des images.

Pour ajouter une image il faut choisir le composant « Item » et lui donner le chemin de référence de l'image dans les propriétés.

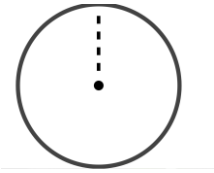
Le code QML généré pour cette application est le suivant (VOIR ANNEXE IV)

Ajout d'un capteur sur la fenêtre

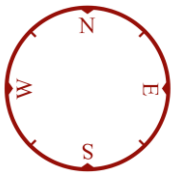
Pour ajouter un capteur, il suffit dans un premier temps de lui donner une image au travers d'un cadran ou autre modélisation. Pour cela il faut d'abord modéliser les différentes images de celui-ci.

Pour un cardant de boussole par exemple il y a deux images :

- La première est la base immobile du cadran et est en arrière-plan

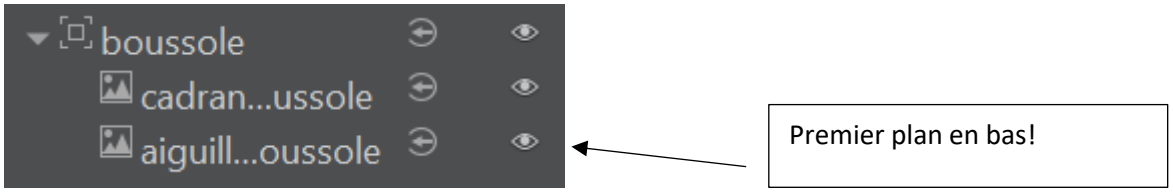


- La deuxième est l'aiguille mobile en premier plan



Attention : il faut bien faire attention à créer des designs vectoriels pour éviter la pixellisation de l'image. De plus prenez soit de les générer sur un fond transparent !

Assurez vous que vos images sur la fenêtre sont dans le bon ordre sur le plan z (c'est-à-dire au niveau des plans).



Animation d'une image

Dans le cas d'une boussole, il va nous falloir tourner son aiguille selon les valeurs récupérées par valcapt. C'est l'unique moment ou il est nécessaire de modifier le programme QML.

La partie à modifier se présente comme ceci :

```
    VALCAPT//appel de valcapt directement dans le programme qml sous le nom de "valeur"
    {
        id:valeur
    }



// Partie à modifier pour ajouter un capteur
    Timer
    {
        property real rot: 0
        id:timer
        interval:100; running:true;repeat: true // on rafraichi l'écran toutes les 0,1 secondes
        onTriggered:
        {
            aiguille_boussole.rotation=valeur.getvalBoussole//animation du cadran de la boussole en influant sur la rotation de son aiguille
            aiguille_accelero.rotation=valeur.getvalAccelero
            aiguille_vent.rotation=valeur.getvalDirVent

            //ajout de capteur : faire évoluer les paramètres du dessins en fonction de valeur.get<nom_val_nouv_capt>
        }

    }
```

aiguille_boussole -> nom de l'image donné à l'aiguille

aiguille_boussole.rotation -> valeur de la rotation de l'image

valeur-> appelle la classe valcapt

valeur.getvalBoussole -> appelle le getter de la dernière valeur de la boussole