16/11/2020

Oscar Prêcheur

MEA5 Polytech Montpellier

FICHE TECHNIQUE

PFE : REPETEUR DE VOL POUR VOILIER

Ce document décrit en détail le fonctionnement général du système de simulation de données capteurs du Dashboard pour voilier

**PARTIE 1 : SIMULATEUR DE DONNEES CAPTEURS**

Table des matières

[Résumé du fonctionnement général 2](#_Toc56436708)

[Descriptif du code 3](#_Toc56436709)

[Classes 4](#_Toc56436710)

[Mainwindow 4](#_Toc56436711)

[serverdistance 10](#_Toc56436712)

[servergite 14](#_Toc56436713)

[servertangage 14](#_Toc56436714)

[servervitesse 14](#_Toc56436715)

# Résumé du fonctionnement général

Ce système est composé de 3 parties :

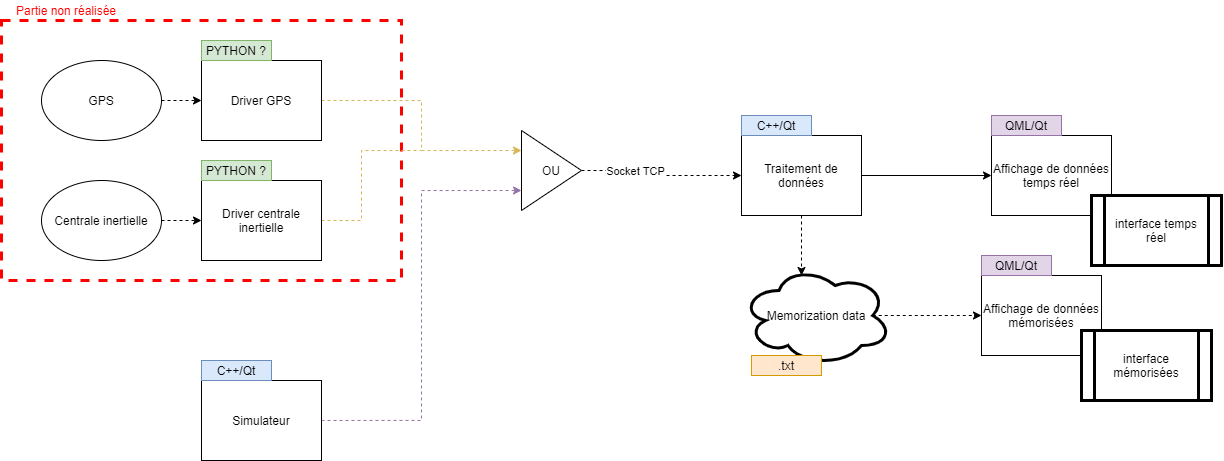
* Acquisition de données
* Traitement de données
* Affichage de données

La partie acquisition de données est effectuée de différentes manière selon l’utilisation du système. D’une part à l’aide de capteurs interfacés avec des drivers, de l’autre avec un simulateur qui les remplace. Dans les deux cas les données sont envoyées sur des sockets TCP et reçu par le programme de traitement des données.

La partie traitement des données lit sur les sockets TCP les valeurs des différentes données capteurs. Il envoie ensuite les valeurs directement au programme d’affichage des données.

La partie affichage des données représente le programme générant l’interface graphique et ses animations.

On peut résumer le fonctionnement du système par ce schéma fonctionnel :



Ce document traite de la partie acquisition de données.

# Descriptif du code

1. **Classes**
   1. Mainwindow
      1. Header
      2. Source
   2. serverdistance
      1. Header
      2. Source
   3. Servergite
      1. Header
      2. Source
   4. servertangage
      1. Header
      2. Source
   5. servervitesse
      1. Header
      2. Source
2. **main.cpp**

## Classes

### Mainwindow

Cette classe permet d’afficher une fenêtre permettant de modifier différentes valeurs capteurs à l’aide de barres coulissantes. Elle permet en plus d’afficher la valeur actuelle de chaque donnée.

#### Header

Mainwindow.h :

**Code :**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QPushButton>

#include <QSlider>

#include <QLCDNumber>

#include <QLabel>

#include <QFont>

#include <QDebug>

#include "servervitesse.h"

#include "servertangage.h"

#include "servergite.h"

QT\_BEGIN\_NAMESPACE

*namespace* **Ui** { *class* **MainWindow**; }

QT\_END\_NAMESPACE

*class* **MainWindow** : *public* QMainWindow

{

Q\_OBJECT

*public*:

*explicit* **MainWindow**(QWidget \*parent = *nullptr*);

~***MainWindow***();

*public* slots:

void **update\_val\_vitesse**();

void **update\_val\_tangage**();

void **update\_val\_gite**();

*private*:

Ui::MainWindow \*ui;

QSlider \*slider\_vitesse;

QSlider \*slider\_tangage;

QSlider \*slider\_gite;

QLabel \*label\_vitesse;

QLabel \*label\_tangage;

QLabel \*label\_gite;

QLCDNumber \*LCD\_vitesse;

QLCDNumber \*LCD\_tangage;

QLCDNumber \*LCD\_gite;

servervitesse \*ValVitesse;

servertangage \*ValTangage;

servergite \*ValGite;

int val\_vitesse=0;

int val\_tangage=0;

int val\_gite=0;

};

#endif *//* *MAINWINDOW\_H*

**Description :**

#include <QMainWindow> -> Permet la création de fenêtres

#include <QPushButton> -> Permet la création de widget bouton

#include <QSlider> -> Permet la création de barres coulissantes

#include <QLCDNumber> -> Permet la dréation de widgets capables d’afficher des valeurs numériques

#include <QLabel> -> Permet la création de zone de texte

#include <QFont> -> Permet d’ajouter des polices de caractères

#include <QDebug> -> Permet d’afficher des données sur terminal

#include "servervitesse.h" -> Appelle la classe servervitesse

#include "servertangage.h" -> Appelle la classe servertangage

#include "servergite.h" -> Appelle la classe servergite

Ces includes appellent les classes utilisées dans la classe Mainwindow.

QT\_BEGIN\_NAMESPACE

*namespace* **Ui** { *class* **MainWindow**; }

QT\_END\_NAMESPACE

Nomination de la classe MainWindow

*public*:

*explicit* **MainWindow**(QWidget \*parent = *nullptr*);

~***MainWindow***();

Builder et destructeur de classe

*public* slots:

void **update\_val\_vitesse**();

void **update\_val\_tangage**();

void **update\_val\_gite**();

Les slots sont des fonctions appelées à un certain moment dans le programme suivant l’état de signaux. Dans ce cas la ces slots permettent de mettre à jour les valeurs des données capteurs.

QSlider \*slider\_vitesse;

QSlider \*slider\_tangage;

QSlider \*slider\_gite;

Ajout des barres coulissantes

QLabel \*label\_vitesse;

QLabel \*label\_tangage;

QLabel \*label\_gite;

Ajout des zones de texte

QLCDNumber \*LCD\_vitesse;

QLCDNumber \*LCD\_tangage;

QLCDNumber \*LCD\_gite;

Ajout des afficheurs des valeurs numériques

servervitesse \*ValVitesse;

servertangage \*ValTangage;

servergite \*ValGite;

Valeurs capteurs récupérées

int val\_vitesse=0;

int val\_tangage=0;

int val\_gite=0;

Valeurs numérique capteurs

#### Sources

Mainwindow.cpp :

**Code :**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(parent)

, ui(*new* Ui::MainWindow)

{

ui->setupUi(*this*);

*//Envoie* *de* *données*

ValTangage = *new* servertangage(65430,*this*);

ValGite = *new* servergite(65431,*this*);

ValVitesse = *new* servervitesse(65432,*this*);

*//Simu* *vitesse*

label\_vitesse = *new* QLabel("Vitesse",*this*);

label\_vitesse->setGeometry(0, 0,50,50);

slider\_vitesse = *new* QSlider(Qt::*Horizontal*, *this*);

slider\_vitesse->setGeometry(0, 50, 300, 40);

slider\_vitesse->setMinimum(0);

slider\_vitesse->setMaximum(120);

LCD\_vitesse = *new* QLCDNumber(*this*);

LCD\_vitesse->setSegmentStyle(QLCDNumber::*Flat*);

LCD\_vitesse->move(200, 20);

QObject::connect(slider\_vitesse, SIGNAL(valueChanged(int)), LCD\_vitesse, SLOT(display(int))) ;

QObject::connect(slider\_vitesse, SIGNAL(valueChanged(int)), *this*, SLOT(update\_val\_vitesse())) ;

*//Simu* *tangage*

label\_tangage = *new* QLabel("Tangage",*this*);

label\_tangage->setGeometry(0, 100,50,50);

slider\_tangage = *new* QSlider(Qt::*Horizontal*, *this*);

slider\_tangage->setGeometry(0, 150, 300, 40);

slider\_tangage->setMinimum(-180);

slider\_tangage->setMaximum(180);

LCD\_tangage = *new* QLCDNumber(*this*);

LCD\_tangage->setSegmentStyle(QLCDNumber::*Flat*);

LCD\_tangage->move(200, 120);

QObject::connect(slider\_tangage, SIGNAL(valueChanged(int)), LCD\_tangage, SLOT(display(int))) ;

QObject::connect(slider\_tangage, SIGNAL(valueChanged(int)), *this*, SLOT(update\_val\_tangage())) ;

*//Simu* *gite*

label\_gite = *new* QLabel("Gite",*this*);

label\_gite->setGeometry(0, 200,50,50);

slider\_gite = *new* QSlider(Qt::*Horizontal*, *this*);

slider\_gite->setGeometry(0, 250, 300, 40);

slider\_gite->setMinimum(-180);

slider\_gite->setMaximum(180);

LCD\_gite = *new* QLCDNumber(*this*);

LCD\_gite->setSegmentStyle(QLCDNumber::*Flat*);

LCD\_gite->move(200, 220);

QObject::connect(slider\_gite, SIGNAL(valueChanged(int)), LCD\_gite, SLOT(display(int))) ;

QObject::connect(slider\_gite, SIGNAL(valueChanged(int)), *this*, SLOT(update\_val\_gite())) ;

*//envoie* *des* *valeurs*

connect(slider\_vitesse, SIGNAL(valueChanged(int)), ValVitesse, SLOT(update(int))) ;

connect(slider\_tangage, SIGNAL(valueChanged(int)), ValTangage, SLOT(update(int))) ;

connect(slider\_gite, SIGNAL(valueChanged(int)), ValGite, SLOT(update(int))) ;

}

void MainWindow::**update\_val\_vitesse**()

{

val\_vitesse=slider\_vitesse->value();

qDebug()<<"VITESSE "<<val\_vitesse;

}

void MainWindow::**update\_val\_tangage**()

{

val\_tangage=slider\_tangage->value();

qDebug()<<"TANGAGE "<<val\_tangage;

}

void MainWindow::**update\_val\_gite**()

{

val\_gite=slider\_gite->value();

qDebug()<<"GITE "<<val\_gite;

}

MainWindow::~***MainWindow***()

{

*delete* ui;

}

**Description :**

La construction de chaque kit pour chaque donnée étant similairement la même nous prendrons comme exemple celle de la donnée de tangage.

ValTangage = *new* servertangage(65430,*this*);

Création du pointeur ValTangage avec servertangage(int numero\_port\_TCP,parent)

label\_tangage = *new* QLabel("Tangage",*this*);

label\_tangage->setGeometry(0, 100,50,50);

Ajout du texte “Tangage” avec QLabel(Qstring text, parent) et placement dans la fenêtre

slider\_tangage = *new* QSlider(Qt::*Horizontal*, *this*);

slider\_tangage->setGeometry(0, 150, 300, 40);

slider\_tangage->setMinimum(-180);

slider\_tangage->setMaximum(180);

Ajout de la barre coulissante, paramétrage de son placement, de son minimum (-180)et de son maximum(180)

LCD\_tangage = *new* QLCDNumber(*this*);

LCD\_tangage->setSegmentStyle(QLCDNumber::*Flat*);

LCD\_tangage->move(200, 120);

Ajout de l’affichage numérique, paramétrage de son style visuel et de son placement

QObject::connect(slider\_tangage, SIGNAL(valueChanged(int)), LCD\_tangage, SLOT(display(int))) ;

Connexion du slot d’affichage du LCD\_tangage (display(int) au signal de changement de valeur de la barre coulissante slider\_tangage (valueChanged(int))

QObject::connect(slider\_tangage, SIGNAL(valueChanged(int)), *this*, SLOT(update\_val\_tangage())) ;

Connexion du slot de mise à jour de valeurs capteurs(update\_val\_tangage())au signal de changement de valeur de la barre coulissante slider\_tangage (valueChanged(int))

void MainWindow::**update\_val\_tangage**()

{

val\_tangage=slider\_tangage->value();

qDebug()<<"TANGAGE "<<val\_tangage;

}

Slot de mise à jour des données capteurs de tangage, val\_tagage prend la valeur de la barre coulissante slider\_tangage. On affiche la valeur sur un terminal pour le debogage.

Cette construction est la même pour les autres jeux de valeurs

MainWindow::~***MainWindow***()

{

*delete* ui;

}

Destructeur de la fenêtre

### serverdistance

Cette classe regroupe toutes les fonctions nécessaires pour l’envoie de données correspondantes à la distance. (Cette données est factice et ne servira que pour le développement, la distance sera calculé en temps réel à partir de la vitesse données par le GPS)

#### Header

serverdistance.h :

**Code :**

#ifndef SERVERDISTANCE\_H

#define SERVERDISTANCE\_H

#include <QObject>

#include <QDebug>

#include <QTcpServer>

#include <QTcpSocket>

*class* **serverdistance** : *public* QObject

{

Q\_OBJECT

*public*:

*explicit* **serverdistance**(quint16 port, QObject \*parent=*nullptr*);

*public* slots:

void **newConnection**();

void **bytesWritten**(qint64);

void **update**(int);

*private*:

QTcpServer \*\_server ;

QTcpSocket \*\_socket ;

void **startStreamingData**();

QByteArray a;

};

#endif *//* *SERVERGITE\_H*

**Déscription:**

#include <QObject> -> appelle de la classe QObject permettant la programmation orientée objets

#include <QDebug> -> Permet d’afficher des données sur terminal

#include <QTcpServer> -> Apelle de la classe de création de server TCP

#include <QTcpSocket> -> Apelle de la classe de création de socket TCP

*public*:

*explicit* **serverdistance**(quint16 port, QObject \*parent=*nullptr*);

Builder de la classe, celle-ci prend en paramètre une valeur quint16 qui correspond au numéro de port sur laquelle elle va communiquer

*public* slots:

void **newConnection**();

void **bytesWritten**(qint64);

void **update**(int);

Création des slots newConnection() lance une connexion TCP, bytesWritten lit le nombre de bytes envoyés et retourne cette valeur, update récupère en paramètre la valeur à envoyer et l’écrit sur la socket.

*private*:

QTcpServer \*\_server ;

QTcpSocket \*\_socket ;

void **startStreamingData**();

QByteArray a;

Création des fonctions et variables utiles, startStreamingData() est la fonction orchestrant l’envoie des données. a est une variable tampon des valeurs à envoyer utilisée par updat(int).

#### Source

serverdistance.cpp :

**Code :**

#include "serverdistance.h"

#include<QTimer>

#include<cmath>

serverdistance::**serverdistance**(quint16 port, QObject \*parent):QObject(parent)

{

\_server = *new* QTcpServer(*this*);

connect(\_server, SIGNAL(newConnection()),*this*, SLOT(newConnection()));

*if* (!\_server->listen(QHostAddress::*Any*, port)) {

qDebug() << "Server could not start !" ;

} *else* {

qDebug() << "Server distance started !" << \_server->serverAddress() ;

}

}

void serverdistance::**newConnection**() {

\_socket = \_server->*nextPendingConnection*();

qDebug() << "Someone connected !!";

connect(\_socket, SIGNAL(bytesWritten(qint64)),*this*, SLOT(bytesWritten(qint64)));

startStreamingData() ;

}

void serverdistance::**bytesWritten**(qint64 nb){

qDebug() << "" << nb << " Bytes sent ..." ;

}

void serverdistance::**update**(int newval)

{

float f = (float)newval;

qDebug() <<f;

QByteArray x(*reinterpret\_cast*<*const* char \*>(&f), *sizeof*(f)) ;

a=x;

\_socket->write(a);

\_socket->flush();

}

void serverdistance::**startStreamingData**() *//frequence* *timer* *à* *regler*

{

*auto* timer = *new* QTimer();

timer->setInterval(100);

connect(timer, SIGNAL(timeout()), *this*, SLOT(update()));

timer->start();

}

**Description :**

#include "serverdistance.h"

Appel du header

#include<QTimer>

Appel de la classe QTimer permettant de crée et timer

#include<cmath>

Appel de la calsse cmath permettant de faire des calculs avancée sur les variables

serverdistance::**serverdistance**(quint16 port, QObject \*parent):QObject(parent)

{

\_server = *new* QTcpServer(*this*);

connect(\_server, SIGNAL(newConnection()),*this*, SLOT(newConnection()));

*if* (!\_server->listen(QHostAddress::*Any*, port)) {

qDebug() << "Server could not start !" ;

} *else* {

qDebug() << "Server distance started !" << \_server->serverAddress() ;

}

}

Lorsque la classe est appelée, celle-ci crée un server TCP et lance la connexion à la socket à l’aide du signal newConnection() du server et du slot newConnection() de la classe.

void serverdistance::**newConnection**() {

\_socket = \_server->*nextPendingConnection*();

qDebug() << "Someone connected !!";

connect(\_socket, SIGNAL(bytesWritten(qint64)),*this*, SLOT(bytesWritten(qint64)));

startStreamingData() ;

}

Détection d’une connexion à la socket et lancement de startStreamingData()

void serverdistance::**startStreamingData**() *//frequence* *timer* *à* *regler*

{

*auto* timer = *new* QTimer();

timer->setInterval(100);

connect(timer, SIGNAL(timeout()), *this*, SLOT(update()));

timer->start();

}

Création d’un timer QTimer, paramétrage de son interval à 100ms, connexion du signal timeout du signal, au slot update. Timer->start() permet de lancer le timer. Cela signifie que toute les 100ms update() sera appelée.

void serverdistance::**update**(int newval)

{

float f = (float)newval;

qDebug() <<f;

QByteArray x(*reinterpret\_cast*<*const* char \*>(&f), *sizeof*(f)) ;

a=x;

\_socket->write(a);

\_socket->flush();

}

Récupération de la valeur à envoyer en paramètre. Casting de la variable int en float. Puis on re cast la valeur pour former un mot QByteArray. On écrit dans le socket le message traduit. Flush() est une sécurité pour être sûre que tout le mot à été envoyé.

### servergite

Même construction que serverdistance.

### servertangage

Même construction que serverdistance.

### servervitesse

Même construction que serverdistance.

## Main.cpp

**Code :**

#include "mainwindow.h

#include <QApplication>

int **main**(int argc, char \*argv[])

{

QApplication a(*argc*, argv);

MainWindow w;

w.show();

*return* a.exec();

}

**Déscription :**

#include "mainwindow.h

#include <QApplication>

Appel de la classe mainwindow et de la classe QApplication permettant de créer une applcation sur le PC.

int **main**(int argc, char \*argv[])

{

QApplication a(*argc*, argv);

MainWindow w;

w.show();

*return* a.exec();

}

Création de l’application avec QApplication a(argc, argv). Création de la fenêtre Mainwindow w, affichage de la fenêtre avec w .show() et execution de l’application avec a.exec()