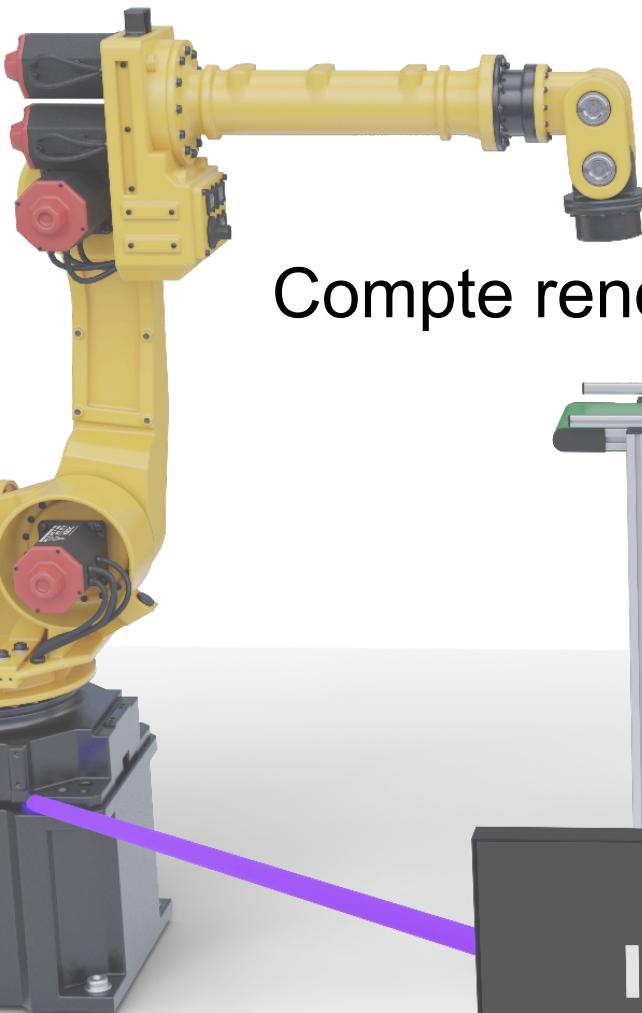
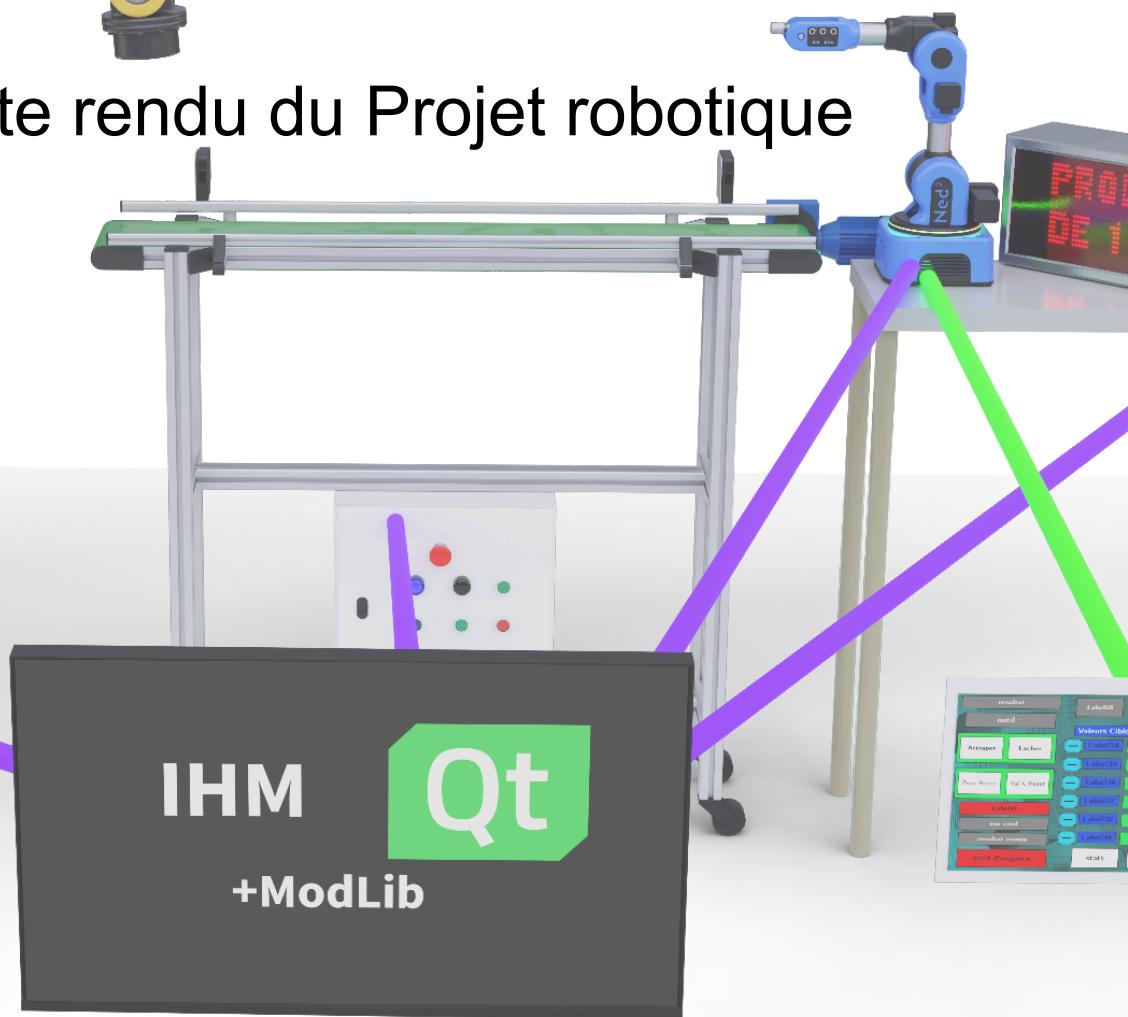


\*\*\*\*\* \*\*\*\*\*  
\*\*\*\*\* \*\*\*  
\*\*\*\*\* \*\*\*\*\*  
\*\*\* \*\*\*\*\*



# Compte rendu du Projet robotique



partie tablette et librairie niryo

# sommaire

<b>sommaire.....</b>	<b>1</b>
<b>Introduction.....</b>	<b>2</b>
<b>1. Présentation générale.....</b>	<b>3</b>
1.1. Le projet dans son ensemble.....	3
1.2. Situation dans le projet.....	4
1.3. Objectifs et cahier des charges.....	5
1.4. Gestion de projet avec trello.....	6
<b>2. Présentation Technique.....</b>	<b>7</b>
<b>2.1. Présentation du matériel et logiciel.....</b>	<b>7</b>
<b>2.2. Configuration de la Tablette.....</b>	<b>8</b>
<b>2.3. Programmation de la tablette.....</b>	<b>10</b>
2.3.1. “teach pendant”.....	10
2.3.2. TPD-703.....	10
2.3.3. HMIWorks.....	11
2.3.4. Développement logiciel.....	12
2.3.5. Solutions Implémentées.....	14
2.3.6. L’interface résultante.....	15
<b>2.4. Librairie Niryo.....</b>	<b>17</b>
2.4.1. QT.....	17
2.4.2. Librairie ModBus.....	17
2.4.3. Classes créées.....	17
<b>Conclusion.....</b>	<b>18</b>
<b>Sitographie.....</b>	<b>19</b>

# Introduction

Dans le cadre de mon BTS Système Numérique Informatique et réseaux, j'ai dû réaliser un projet autour de la programmation et de la robotique avec trois autres étudiants de ma classe. Précisément, \*\*\*\*\* \*\*\*\*, \*\*\* \*\*\*\*\* , \*\*\*\*\* \*\*\*\*\*. Nous devions réaliser la programmation pour un projet en robotique.

Durant ce projet d'une durée de 200 heures, nous avions chacuns des tâches différentes afin de répartir le travail nécessaire pour venir à bout de ce projet.

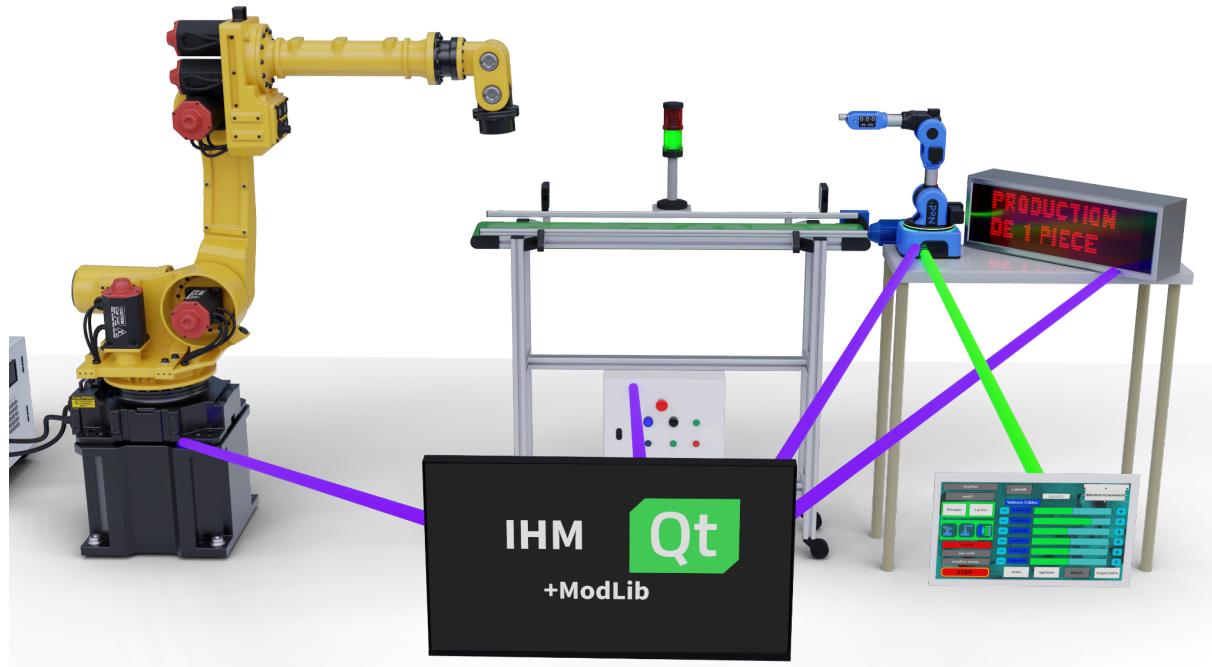
Dans ce dossier de projet, je vais commencer par présenter le projet dans son ensemble, puis je vais expliquer ma part dans le projet et mon cahier des charges.

Je continuerai par expliquer ce que je devais réaliser, avant d'expliquer mes solutions et réalisations.

Finalement je conclurai sur le projet et les difficultés rencontrées durant le projet.

# 1. Présentation générale

## 1.1. Le projet dans son ensemble



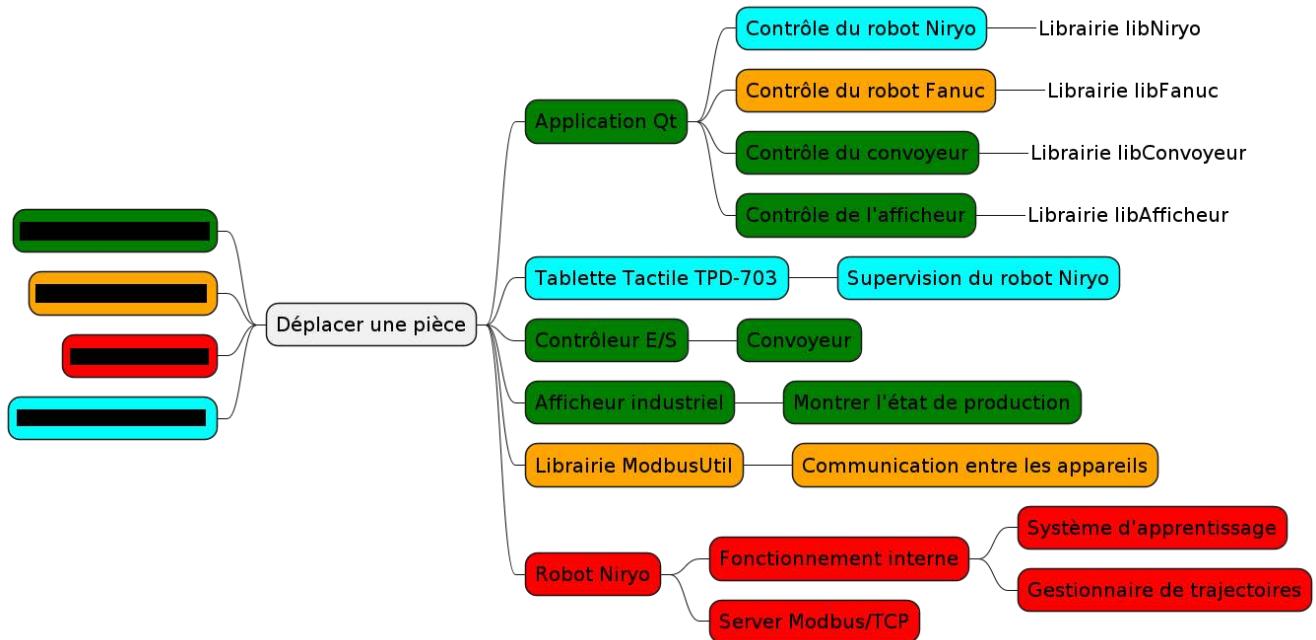
Le projet consiste à utiliser deux bras robotique et un tapis roulant afin de simuler une chaîne de production simplifiée par rapport à une vraie chaîne de production. L'ensemble de ces appareils et un afficheur, pour montrer l'état de la production, sont contrôlés via une IHM (Interface Homme Machine) contrôlant la production.

La chaîne de production consiste à utiliser le Fanuc (en jaune) pour récupérer un objet, le déposer sur le tapis roulant. Le tapis roulant transporte l'objet et détecte la position de l'objet avec deux capteurs. Quand l'objet arrive en bout de tapis roulant, le robot Niryo ned 2 (en bleu), le récupère et le dépose ailleurs.

L'IHM contrôlant la production ainsi que la tablette tactile contrôlent les autres appareils via le protocole Modbus/TCP. Dans le schéma ci-dessus, les communications de l'IHM sont représentées en bleu et la communication entre la tablette et le Niryo est en vert .

Modbus/TCP est un protocole de communication pour des appareils simples permettant de transporter peu de données afin de contrôler des appareils par exemple des robots ou des appareils industriels

## 1.2. Situation dans le projet



Ce schéma représente le travail devant être réalisé par chacun des membres du projet grâce à un code couleur. J'ai pour ma part réalisé les parties en cyan, notamment la librairie permettant à l'application QT de contrôler le niryo ned2 et la programmation de la tablette servant à superviser ce même robot.

### 1.3. Objectifs et cahier des charges



Dans ce projet j'ai eu 2 Objectifs :

- Le premier était d'utiliser une tablette tactile afin de créer une interface permettant de contrôler le robot Niryo ned2
- Le deuxième était de créer une librairie contenant toutes les fonctions nécessaires pour contrôler le ned2 depuis le logiciel de contrôle (IHM)

Durant ce projet j'ai eu un cahier des charges à respecter pour chacun des 2 objectifs que j'ai eus à réaliser.

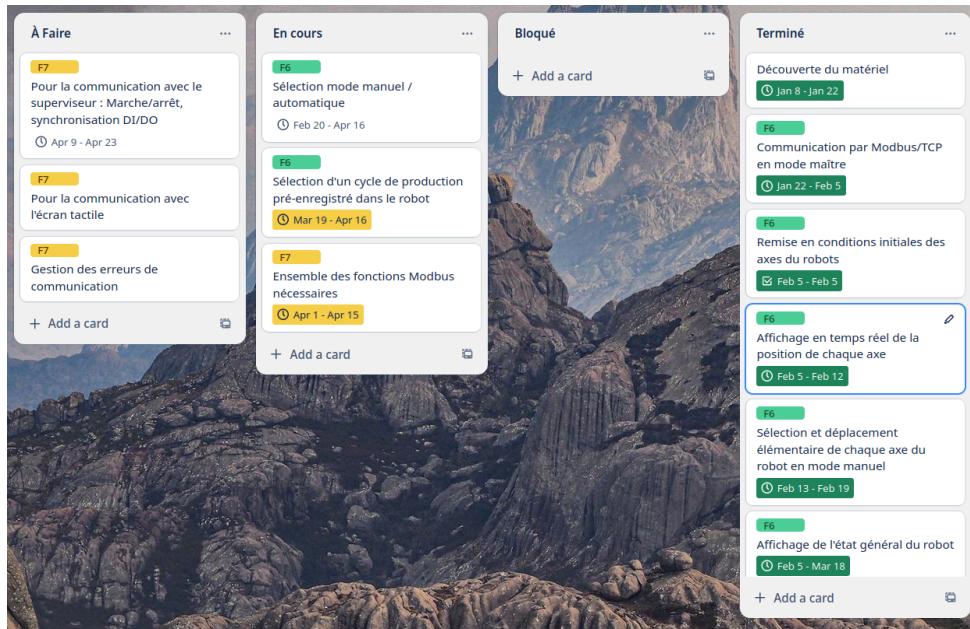
Pour le premier, le pilotage du Niryo grâce à un écran tactile, je devais :

- Sélection mode manuel / automatique
- Remise en conditions initiales des axes du robots
- Sélection et déplacement élémentaire de chaque axe du robot en mode manuel
- Sélection d'un cycle de production pré-enregistré dans le robot
- Affichage de l'état général du robot
- Affichage en temps réel de la position de chaque axe
- Communication par Modbus/TCP en mode maître

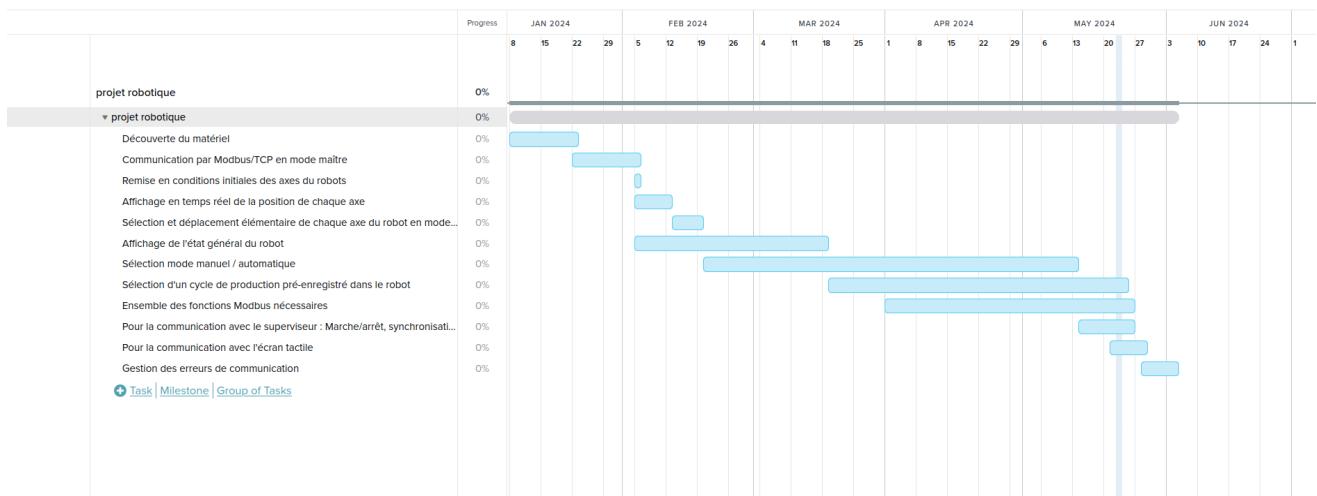
Pour le deuxième, la librairie de communication Modbus/TCP entre le Niryo et l'IHM, je devais:

- Pour la communication avec le superviseur : Marche/arrêt, synchronisation DI/DO
- Pour la communication avec l'écran tactile
- Ensemble des fonctions Modbus nécessaires
- Gestion des erreurs de communication

## 1.4. Gestion de projet avec trello



Pour simplifier la gestion du projet nous avons utilisé le site trello. Ce site nous à permis de noter les dates de début et de fin de chaque tâche qui ont par la suite permis de générer un diagramme de Gantt du projet complet.



## 2. Présentation Technique

### 2.1. Présentation du matériel et logiciel



HMIWorks est un logiciel créé par ICP DAS pour permettre la programmation de leurs tablettes tactiles et de leurs afficheurs industriels en C et en Ladder.

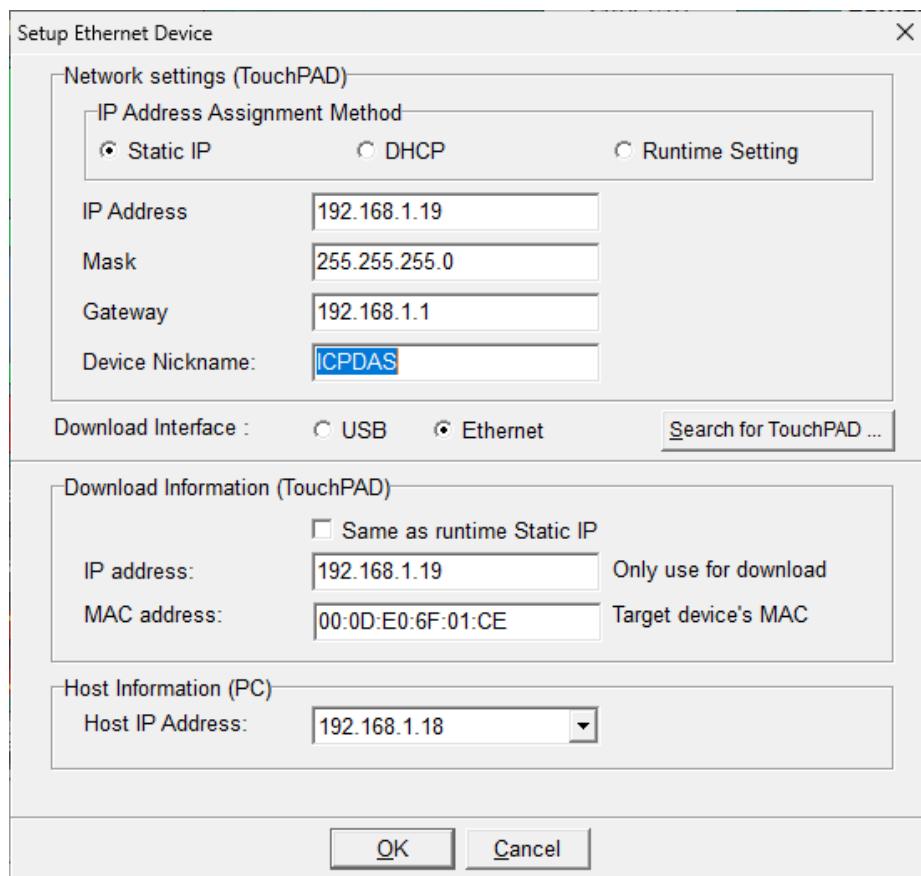
tablette TPD-703



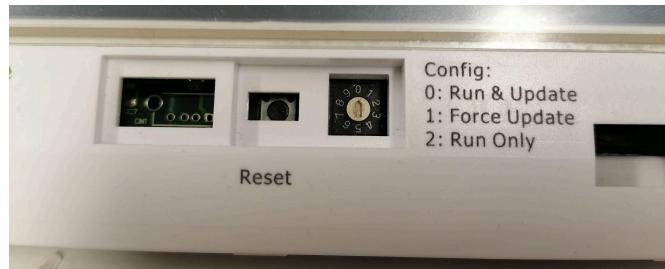
C'est une tablette tactile programmable construite par ICP DAS. Celle-ci possède un écran tactile résistif de 7 pouces avec une résolution de 800 par 480 pixels. Elle est alimentée en PoE (Power over Ethernet). Et possède 16 Méga octets de stockage en SDRAM.

Elle est par exemple capable de contrôler des machines et appareils industriels et possède du code pour communiquer en modbus/TCP.

## 2.2. Configuration de la Tablette



Avant de pouvoir programmer la tablette j'ai dû commencer par configurer l'adresse IP de la tablette. Cela est fait en utilisant le menu ci-dessus et en mettant la tablette en mode "force update". Ceci est réalisé grâce à l'interrupteur rotatif derrière la plaque en plastique de façade de la tablette.



J'ai d'abord essayé de configurer la tablette en étant sur le réseau du lycée cependant la tablette récupérait les informations du serveur DHCP du lycée. Après quoi j'ai conçu un serveur DHCP et un réseau local, en effet une fois l'élaboration du projet terminé tous les appareils seront placés dans un réseau local. Mais même avec un dhcp locale le logiciel HMIWorks n'arrivait pas à téléverser le programme dans la tablette. J'ai finalement réglé le problème en configurant le DHCP pour refuser de donner une adresse à la tablette. Ce dernier changement à suffit pour rendre la connexion entre le logiciel de programmation et la tablette.

## 2.3. Programmation de la tablette

### 2.3.1. "teach pendant"



//présentation de la tablette et "teach pendant"

Afin de pouvoir contrôler un robot dans un environnement de production, il est utile d'avoir une télécommande afin de pouvoir créer des trajectoires, ou remettre le robot à sa position de départ. C'est pour cette raison que les robots industriels ont un appareil appelé un "teach pendant" qui permet de ne pas avoir à utiliser le logiciel du robot pour la moindre opération.

### 2.3.2. TPD-703



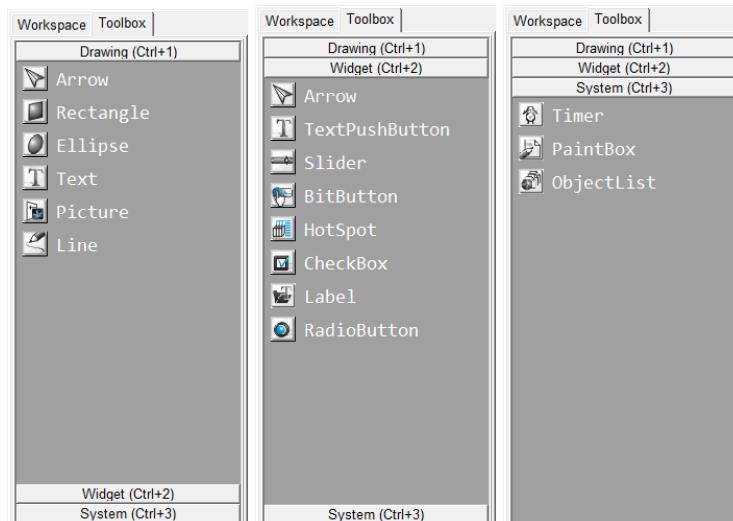
Cependant le Niryo ned2 étant un robot d'apprentissage, aucun appareil de ce genre n'existe pour ce dernier. C'est pour cela que ma première tâche dans ce projet consistait à ajouter une tablette tactile programmable appelée TPD-703. Elle sera utilisée pour simuler un "teach pendant" pour le Niryo ned 2

### 2.3.3. HMIWorks



Cette tablette possède un logiciel pour la programmer, trouvable sur le site du fabricant (ICPDAS). Ce logiciel nommé HMIWorks permet de configurer l'adres IP de la Tablette, et génère tous les fichiers qui contiennent le programme par défaut de la tablette. Il donne aussi la possibilité de définir des écrans pour la tablette qui se traduiront par des fichiers de définition(.h) dans le dossier qui contient les programmes.

Le logiciel HMIWorks génère beaucoup de fichiers nécessaires au bon fonctionnement de la tablette et même les fichiers "makefile" qui sont des fichiers permettant de contrôler la compilation du programme.



Ce logiciel permet de créer beaucoup de boutons, sliders et étiquettes pour interagir avec la tablette

Ce logiciel permet de créer des boutons et étiquettes pré -faites.

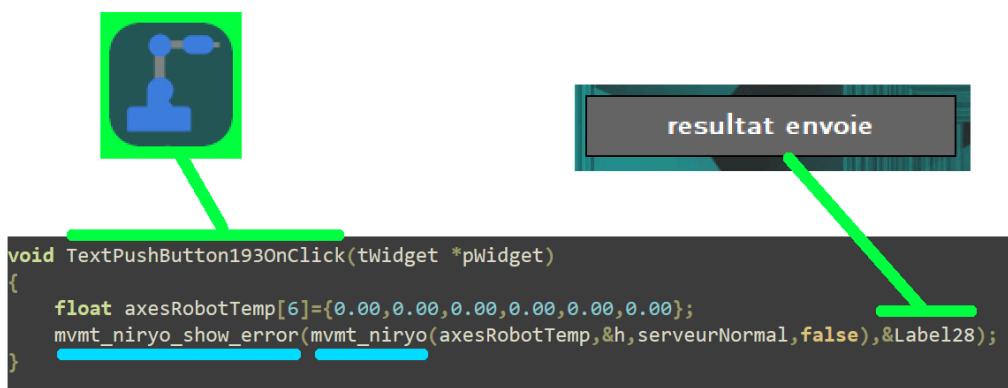
Lors de la création d'un bouton, le logiciel va définir le bouton dans les fichiers du projet, puis on peut rajouter le nom d'une fonction à exécuter lors de l'appui sur le bouton. Cette fonction programmer en C donne des fonctionnalités à ce bouton.

### 2.3.4. Développement logiciel

Afin de créer un programme capable de répondre aux objectifs que l'on me donnés avec la tablette, j'ai décidé de créer des fichiers de librairie qui me permettent de créer des fonctions pour des utilisations spécifiques. Par exemple, niryo.h contient une fonction capable de lancer un déplacement sur le Niryo a partir d'un tableau de valeur contenant les angles de chaque axe.

Cette décision m'a permis de simplifier mon programme et de ne pas refaire le même code plusieurs fois.

Voici l'exemple du fonctionnement du bouton de remise à zéro:



Lors de l'appui sur le bouton de remise à zéro la tablette va appeler la fonction TextPushButton193OnClick. Cette fonction va commencer par créer le tableau de valeur ici contenant uniquement des 0.00, afin de remettre les 6 axes du robot à zéro.

Après ça la fonction du bouton appelle une fonction du fichier Niryo appeler **mvmt\_niryo**, avant de rediriger le code d'erreur renvoyé par la fonction **mvmt\_niryo** vers **mvmt\_niryo\_show\_error**.

Cette dernière fonction va prendre le code d'erreur de **mvmt\_niryo** sous la forme d'un nombre entier. Puis finalement elle va renvoyer le texte associé à ce code d'erreur dans l'étiquette appelé Label28 qui sera affiché sur l'écran.

```
int mvmt_niryo(float _axesRobotcible[6],tHandle *_h,int serverID,bool HardReset){
    //envoie des axes
    for( int i = 0;i<6;i++){//
        ecritureFNCT06 = RadiantToAxes(_axesRobotcible[i]);
        if(!EcritureFNCT06W(*_h,i)){
            return 1;
        }
        hmi_DelayUS(500);
    }

    //execution cmd
    if (demandeDeMouvementNiryo(*_h,100,false,false) != 0){
        return 2;
    }

    //reinitialisation de la connexion
    if(HardReset){
        if(!TCPHardReset(_h,serverID)){
            return 3;
        }
    }

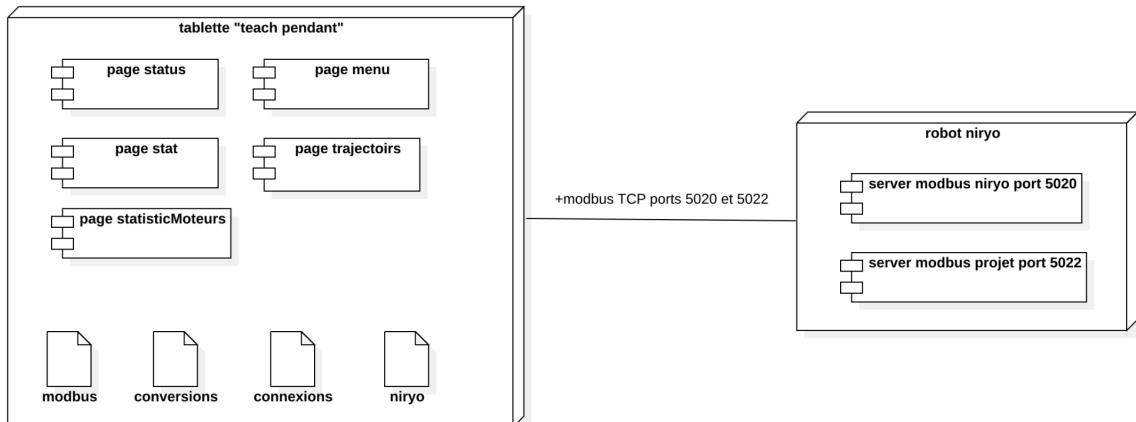
    return 0;
}
```

La fonction **mvmt\_niryo** prend comme argument : un tableau de nombres à virgule flottante , un tHandle qui permet à la tablette de différencier les connexions TCP, un id qui permet en cas de reconnexion de se connecter au bon serveur car 2 seront utilisés dans la communication avec le Niryo et une variable booléen. Cette dernière variable permet de réinitialiser la connexion après l'envoi de la commande, car certaines commandes bloquent l'exécution du serveur Modbus/TCP du robot. Ce qui empêche le robot de répondre aux demandes de la tablette pendant plusieurs secondes.

Cette fonction commence par prendre les valeurs du tableau `_axeRobotcible` et les envoient au Niryo aux adresses 0 à 6 du serveur préexistant du Niryo. Une fois cela fait la fonction `demandeDeMouvementNiryo` est appellé. Elle permet de vérifier si le robot n'est pas déjà en train d'exécuter un mouvement et si le robot n'a pas détecté de collision avant d'envoyer le signal de lancer un mouvement.

Finalement si `HardReset` est vraie la connexion tcp est terminé et immédiatement recréée, ce n'est utile que dans certains cas et contourne un défaut du Niryo ned2

### 2.3.5. Solutions Implémentées



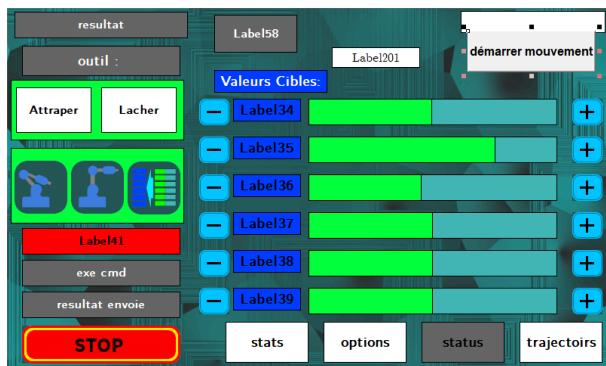
En fin de projet j'ai fini par atteindre une structure de programme composée de fichiers de définitions, qui concentrent la majorité des fonctions nécessaires au bon fonctionnement des différents menus.

Ces fichiers étaient au nombre de 4 :

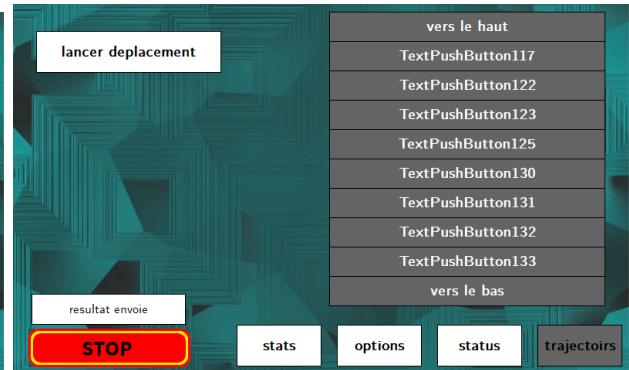
- Le fichier **modbus** redéfinit les fonctions Modbus/TCP de la tablette pour simplifier leurs utilisations.
- Le fichier **conversions** crée des fonctions dont certaines permettent de convertir les données venant du robot vers un format utilisable dans des variables, et d'autres permettent de convertir des radians en degrés et vice versa.
- **connexions** définit les fonctions de gestion de la connexion au Niryo en se servant de certaines des fonctions préexistantes dans la tablette.
- **niryo** est le fichier contenant les fonctions les plus complexes et s'appuie sur toutes les fonctions définies dans les fichiers précédents.

### 2.3.6. L'interface résultante

L'interface finale que j'ai créée pour la tablette est divisée en 5 écrans principaux avec chacun un rôle spécifique :



écran de contrôle



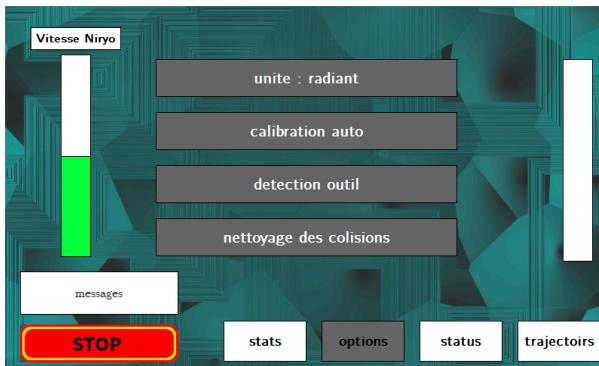
écran de trajectoire

- **L'écran de control du robot ned2 :**

Il permet d'afficher en temps réel l'angle de chacun des axes du robot, de lancer un mouvement et de contrôler l'outil.

- **L'écran des trajectoires :**

Il montre toutes les trajectoires existantes sur le robot et permet de les lancer. Toutes les fonctionnalités de cet écran s'appuient sur le second serveur Modbus/TCP créé par \*\*\*\*\*.



écran menu



écran des statistiques

- **L'écran menu :**

sert à modifier des paramètres importants comme la vitesse des moteurs du Niryo ou de lancer une calibration.

- **L'écran des statistiques :**

donne des informations diverses mais non cruciales sur le robot.

	voltage	temperature	Code d'erreur 0 = OK	
J1-Base	LabelMotors	LabelMotors	LabelMotors	
J2-Shoulder	LabelMotors	LabelMotors	LabelMotors	
J3-Elbow	LabelMotors	LabelMotors	LabelMotors	
J4-Forearm	LabelMotors	LabelMotors	LabelMotors	
J5-Wrist	LabelMotors	LabelMotors	LabelMotors	
J6-Hand	LabelMotors	LabelMotors	LabelMotors	
End Effector	LabelMotors	LabelMotors	LabelMotors	
Tool	LabelMotors	LabelMotors	LabelMotors	retour

écran d'information des moteurs

- **L'écran d'information des moteurs** : celui-ci affiche les données des moteurs rendues accessibles grâce aux modifications faites par \*\*\*\*\* sur le robot.

## 2.4. Librairie Niryo

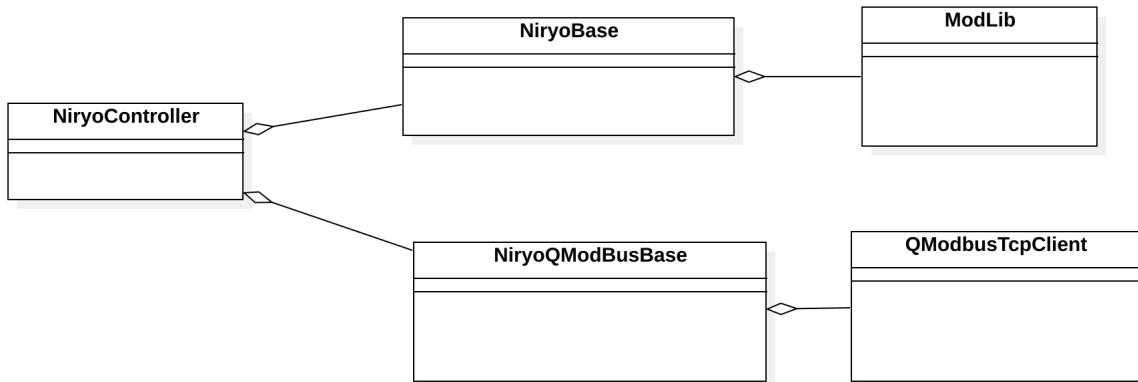
### 2.4.1. QT

QT est un environnement de programmation qui permet de créer un logiciel et contient des librairies et permettant de créer un logiciel graphique sans avoir à recréer toutes les fonctionnalités de base.

### 2.4.2. Librairie ModBus

Lors de la programmation de la librairie QT pour contrôler le robot Niryo, j'ai eu à choisir entre la librairie ModLib créée par \*\*\* et la librairie QModBusTcp fournie par QT. Finalement j'ai décidé d'utiliser QModBusTcp pour simplifier la programmation de ma librairie et car cette dernière possède une meilleure gestion des erreurs.

### 2.4.3. Classes créées



Pour ma librairie QT, j'ai créé 2 classes. la plus importante est la classe NiryoController qui permet de contrôler et de récupérer les informations provenant du robot. tandis que la classe NiryoBase ou NiryoQmodBusBase simplifie la communication au Niryo. Sur le schéma ci-dessus j'ai décidé de représenter les deux possibilités pour les classes NiryoBase et NiryoQModBusBase. Mais lors de la réalisation, seule la classe NiryoQModBusBase basée sur QModBusTcpClient sera utilisée car ModLib serait trop compliquée à implémenter.

# Conclusion

Durant ce projet j'ai rencontré des complications qui m'ont permis de voir les difficultés qui peuvent apparaître dans ce type de projet :

Par exemple, j'ai dû m'habituer à la programmation en C pour programmer la tablette. un langage de programmation qui ne possède pas certaines des fonctionnalités des langages comme le C++ qui à été vu en cours. Le C ne possède notamment pas de concept de classe qui permet de rendre les programmes plus compréhensibles et évite les répétitions.

Aussi le logiciel HMIWorks possèdent des documentation un peu mal traduit. Et ne fonctionnent que sur windows ce signifie que j'ai dû passé par une machine virtuelle pour l'utiliser.

Et finalement, le niryo avait un bug lors du lancement des mouvements. qui empêchait le robot de répondre durant un mouvement. qui à été réglé par \*\*\*\*\*.

Cela étant dit, j'ai tout de même atteint beaucoup des objectifs imposés par le cahier des charges.

Notamment, la tablette remplit son rôle de supervision, de contrôle du niryo du Niryo ned2 et est capable de lancer des trajectoire.

Tandis que la librairie pour contrôler le niryo est bien débuter et devrait être finit dans les temps.

# Sitographie

documentation Niryo:

[https://pymodbustcp.readthedocs.io/en/latest/package/class\\_ModbusServer.html](https://pymodbustcp.readthedocs.io/en/latest/package/class_ModbusServer.html)

<https://docs.niryo.com/api/modbus/use-the-modbus-tcp-server/>

<https://archive-docs.niryo.com/dev/ros/v4.1.1/en/source/modbus.html>

faq icpdas <https://www.icpdas.com/en/faq/index.php?page=1&kind=329#882>

documentation HMIWorks:

manuel HMIWorks

[https://www.icpdas.com/web/product/download/software/development\\_tool/hmiworks/document/manual/HMIWorks\\_Software\\_User\\_Manual\\_en\\_v1.5.pdf](https://www.icpdas.com/web/product/download/software/development_tool/hmiworks/document/manual/HMIWorks_Software_User_Manual_en_v1.5.pdf)

Référence de programmation

[https://www.icpdas.com/web/product/download/software/development\\_tool/hmiworks/document/manual/HMIWorks\\_API\\_Reference\\_en\\_v1.7.5.pdf](https://www.icpdas.com/web/product/download/software/development_tool/hmiworks/document/manual/HMIWorks_API_Reference_en_v1.7.5.pdf)

tutoriel / guide de programmation

[https://www.icpdas.com/web/product/download/software/development\\_tool/hmiworks/document/manual/TouchPAD\\_Programming\\_Guide\\_en.pdf](https://www.icpdas.com/web/product/download/software/development_tool/hmiworks/document/manual/TouchPAD_Programming_Guide_en.pdf)

## Table of Contents

Début de connexion.h.....	1
Début de conversion.h.....	3
Début de niryo.h.....	7
Début de modbus.h.....	17
Début de __Connexion.h.....	24
Début de __Statut.h.....	26
Début de __Trajectoire.h.....	41
Début de __Statistic.h.....	48
Début de __StatisticMoteurs.h.....	51
Début de __Menu.h.....	54
Début de __Clavier.h.....	57

## Début de connexion.h

```
/*
    system de connection pour la tablette

    checkConnexion() permet de verifier la connexion tcp au robot
    et renvoi
    0 si tout se passe comme prevu
    1 si id server est mauvais
    2 si le nombre d'essaie est dépasser

    connexionModBus() permet de se connecter au robot
    et renvoi
    les meme erreur que checkConnexion()
```

18/03/2024

```
*/
#ifndef CONNECTIONS_H
#define CONNECTIONS_H
int server1DisconnectCount = 0;
int server2DisconnectCount = 0;

int connexionModBus(tHandle *_h,int serverID){//serverID normal = 1 custom = 2
    int connexionStatus = 0;
    int state = -1;

    state = hmi_TCPState (*_h);
```

```

if(state == -1){return 5;}
if(state == 3){return 4;connexionStatus=2;}
if(connexionStatus == 0{
    if(serverID == 1){
        *_h = mtm_Register(1, TCP_IPADDR(192,168,1,10), 5020);// connexion au
serveur du niryo
    } else if(serverID == 2){
        *_h = mtm_Register(1, TCP_IPADDR(192,168,1,10), 5022);// connexion au
serveur pour le projet
    } else {
        return 1;
    }
    connexionStatus = 1;
}
int loopCount = 0;

while(connexionStatus == 1){// tant que connexion en cours

    hmi_DelayUS(10000);
    state = hmi_TCPState (*_h);
    if (state == -1){
    } else if(state == 0){
    } else if(state == 1){
    } else if(state == 2){
    } else if(state == 3){
        connexionStatus = 2;
    }
    loopCount = loopCount + 1;
    if(loopCount >= 200){
        connexionStatus = 0;
        mtm_Unregister(*_h);
        *_h = -1;
        return 2;
    }
}
if(connexionStatus == 2){// si connexion reussi

}
return 0;
}

int checkConnexion(tHandle *_h,int serverID){//serverID normal = 1 custom = 2
//partie verification de l etat de la connection tcp
int state = -1;
state = hmi_TCPState (*_h);
if (state == -1){
    mtm_Unregister(*_h);
    connexionModBus(_h,serverID);
} else if(state == 0){//idle
    if(serverID == 1){server1DisconnectCount++;}
}
}

```

```

        else if(serverID == 2){server2DisconnectCount++;}
        else {return 1;}
    } else if(state == 1){//listen
        if(serverID == 1){server1DisconnectCount++;}
        else if(serverID == 2){server2DisconnectCount++;}
        else {return 1;}
    } else if(state == 2){//connecting

    } else if(state == 3){//connected
        if(serverID == 1){server1DisconnectCount = 0;}
        else if(serverID == 2){server2DisconnectCount = 0;}
        else {return 1;}
    }
    //partie reconnection si necessaire
    int erreur = 0;
    if((serverID == 1 ? server1DisconnectCount : server2DisconnectCount) > 10 && state != 3)
{
    mtm_Unregister(*_h);
    *_h = -1;
    erreur = connexionModBus(_h,serverID);
}

return erreur;
}

```

bool TCPHardReset(tHandle \*\_h,int serverID){//fonction pour regler les perte de connection lors du lancement de trajectoire

//termine a connexion pour en recreer une autre

```

    mtm_Unregister(*_h);
    *_h = -1;
    connexionModBus(_h,serverID);
    return true;
}

```

#endif

Fin de connexion.h

## **Début de conversion.h**

```

#ifndef CONVERSION_H
#define CONVERSION_H

```

```
//definition de pi car il n est pas definit  
#define PI 3.1415926535897932384626433832795028841971693993751058209749445923078164
```

```
//////////  
// conversion de connées  
//////////
```

```
int customDataToInt(WORD w){//conversion des voltages moteurs
```

```
    return (((int)w)/1000);
```

```
}
```

```
float customDataToFloat(WORD w){//conversion des voltages moteurs
```

```
    return (((float)w)/1000);
```

```
}
```

```
signed short axesToSignedShort(WORD w){// transforme les données d'axes du robot en Signed short
```

```
    signed short result = -1;
```

```
    if ((w >> 15) == 1){
```

```
        result = 0-(signed short)(w & 0x7fff);
```

```
    } else {
```

```
        result = (signed short)(w & 0x7fff);
```

```
    }
```

```
    return result;
```

```
}
```

```
int axesToInt(WORD w){// transforme les données d'axes du robot en Int
```

```
    int result = -1;
```

```
    if ((w >> 15) == 1){
```

```
        result = 0-(int)(w & 0x7fff);
```

```
    } else {  
        result = (int)(w & 0x7fff);  
    }  
    return result;  
}
```

float axesToFloat(WORD w){// transforme les données d'axes du robot en float

```
    float result = -1;  
    if ((w >> 15) == 1){  
        result = 0-(float)(w & 0x7fff);  
    } else {  
        result = (float)(w & 0x7fff);  
    }  
    return result;  
}
```

float axesToRadian(WORD w){// transforme les données d'axes du robot en float et en radian

```
    float result = -1;  
    if ((w >> 15) == 1){  
        result = 0-(float)(w & 0x7fff);  
    } else {  
        result = (float)(w & 0x7fff);  
    }  
    result = result / 1000;  
    return result;  
}
```

float axesToDegrees(WORD w){// transforme les données d'axes du robot en float et en degree

```
    float result = -1;  
    if ((w >> 15) == 1){  
        result = 0-(float)(w & 0x7fff);  
    }
```

```
    } else {  
        result = (float)(w & 0x7fff);  
    }  
    result = result / 1000;  
    result = (result*(180/PI));  
    return result;  
}
```

WORD RadiantToAxes(float f){// transforme un float en radiant en donnees pour le robot

```
WORD result = 0;  
f = f*1000;  
if(f < 0){  
    result = (1 << 15) + (((WORD)-f) & 0x7fff);  
} else {  
    result = (((WORD)f) & 0x7fff);  
}  
return result;
```

WORD DegreeToAxes(float f){// transforme un float en degrees en donnees pour le robot

```
WORD result = 0;  
f = (f*(PI/180));  
f = f*1000;  
if(f < 0){  
    result = (1 << 15) - (((unsigned short)f) & 0x7fff);  
} else {  
    result = (((unsigned short)f) & 0x7fff);  
}  
return result;
```

```
}
```

```
float degreeToRadian(float f){// transforme les degre en radiant  
    f = (f*(PI/180));  
    return f;  
}
```

```
float radiantToDegree(float f){// transforme les radiant en degrees  
    f = (f*(180/PI));  
    return f;  
}
```

```
////////////////////////////////////////////////////////////////////////  
// fin de conversion de connées  
////////////////////////////////////////////////////////////////////////  
  
#endif
```

### Fin de conversion.h

## Début de niryo.h

```
/*  
fonction pour simplifier les demande de mouvement au niryo et gérer les erreurs  
  
*/  
#ifndef NIRYO_H  
#define NIRYO_H
```

```
int mvmt_niryo(float _axesRobotcible[6],tHandle *_h,int serverID,bool HardReset){

    //envoie des axes

    for( int i = 0;i<6;i++){//



        ecritureFNCT06 = RadianToAxes(_axesRobotcible[i]);


        if(!EcritureFNCT06W(*_h,i)){


            return 1;

        }

        hmi_DelayUS(500);

    }

    //execution cmd

    if (demandeDeMouvementNiryo(*_h,100,false,false) != 0){


        return 2;

    }

    //reinitialisation de la connexion

    if(HardReset){


        if(!TCPHardReset(_h,serverID)){


            return 3;

        }

    }

    return 0;

}
```

```
bool mvmt_niryo_show_error(int _error,tLabel *L){


    if(_error == 0){


        LabelTextSet(L, "pas d erreur");


    } else if(_error == 1){


        LabelTextSet(L, "erreur d'envoie");


    } else if(_error == 2){
```

```
    LabelTextSet(L, "err lancement mvmt");
} else if(_error == 3){
    LabelTextSet(L, "erreur reset");
}
return true;
}
```

```
bool mvmt_arret(tHandle *_h, tLabel *L){
    bool err = false;

    ecritureFNCT06 = 0xffff;
    err = EcritureFNCT06W(*_h,110);//mvmt

    lectureFNCT03[0] = 0;
    LectureFNCT03R(*_h,110,1);
    if(err && lectureFNCT03[0] > 0){
        LabelTextSet(L,"signal envoyer");
    } else {
        LabelTextSet(L,"signal perdue");
    }
    return true;
}
```

```
bool mvmt_toggleOutil(tHandle *_h, tLabel *L,bool OnOff,int outilEquiper){
    bool error = false;
    int offset = 0;
    if(!OnOff){//si off / false est selectionner
        offset = 1;//change l'adresse pour fermer pince / relacher objet
    }
    if(outilEquiper == 31){
        ecritureFNCT06 = 0xffff;
```

```
error = EcritureFNCT06Wmvmt(*_h,(512+offset));
} else if( outilEquiper == 11 || outilEquiper == 12 || outilEquiper == 13 ){
    ecritureFNCT06 = 0xffff;
    error = EcritureFNCT06Wmvmt(*_h,(511-offset));
}

if(error){
    LabelTextSet(L,"cmd outil envoyer");
} else {
    LabelTextSet(L,"echec cmd outil");
}

return true;
}
```

```
int mvmt_calibration(tHandle *_h,int serverID,bool HardReset){

    //envoie demande calibration
    if (demandeDeMouvementNiryo(*_h,310,true,true) != 0){
        return 1;
    }

    //envoie depart calibration
    if (demandeDeMouvementNiryo(*_h,311,true,true) != 0){
        return 2;
    }

    //reinitialisation de la connexion
    if(HardReset){
        if(!TCPHardReset(_h,serverID)){
            return 3;
        }
    }
}
```

```
    }

    return 0;

}

bool mvmt_calibration_show_error(int _error,tLabel *L){

    if(_error == 0){

        LabelTextSet(L, "calibration demarrer");

    } else if(_error == 1){

        LabelTextSet(L, "erreur de demande");

    } else if(_error == 2){

        LabelTextSet(L, "err de demarrage calibration");

    } else if(_error == 3){

        LabelTextSet(L, "erreur reset");

    }

    return true;

}
```

```
bool mvmt_detect_tool(tHandle *_h,tLabel *L){

    if (demandeDeMouvementNiryo(*_h,500,false,true) != 0){

        LabelTextSet(L, "erreur detection outil");

        return false;

    }

    LabelTextSet(L, "outil mis a jour");

    return true;

}
```

//////// timer //////////

```
typedef struct {

    unsigned long ref_time;

    unsigned long distance_to_ref;
```

```
} timer ;
```

```
timer timer_create(unsigned long distanceMS){  
    timer result;  
    result.ref_time = hmi_GetTickCount();  
    result.distance_to_ref = distanceMS;  
    return result;  
}
```

```
bool timer_isFinished(timer *t){  
    if(hmi_GetTickCount() > (t->ref_time + t->distance_to_ref) || hmi_GetTickCount() < (t->ref_time - t->distance_to_ref)){  
        return true;  
    } else {  
        return false;  
    }  
}
```

```
void timer_reset(timer *t){  
    t->ref_time = hmi_GetTickCount();  
}
```

////////// trajectory //////////

```
typedef struct {  
    int ID;  
    char name[21];  
    int index;  
} trajectory ;
```

```

int traj_indexation(trajectory t[],int sizeOfTArray,tHandle _h){
    if(!LectureFNCT03R(_h,50,1)){return 1;}

    int nbTraj = (int)lectureFNCT03[0];
    if(nbTraj < 1 || nbTraj > sizeOfTArray){return 2;}

    if(!LectureFNCT03R(_h,0,1)){return 1;}
    if(lectureFNCT03[0] > 0){return 5;}

    for(int i = 0 ; i < nbTraj ; i++){
        ecritureFNCT06 = (WORD)i;
        if(!EcritureFNCT06W(_h,41)){return 3;} // selection traj. i sur le robot
        int loopcount = 0;
        do{
            hmi_DelayUS(500);
            LectureFNCT03R(_h,51,1);
            loopcount++;
        } while(lectureFNCT03[0] == 0 && loopcount < 100); //tant que pas de traj.
        selectionner
        if(loopcount > 100){return 4;}
        if(!LectureFNCT03R(_h,10,22)){return 1;}

        // mise des valeur dans le tableau de traj.
        t[i].index = i;
        for(int j = 0 ; j < 20 ; j++){ // mise du nom dans le tableau (20 char)
            t[i].name[j] = (char)lectureFNCT03[j];
        }
        t[i].name[20] = '\0'; // termination du nom
        t[i].ID = (int)lectureFNCT03[21];
    }
}

```

```

ecritureFNCT06 = 0x00;
if(!EcritureFNCT06W(_h,51)){return 3;}//desectionne la trajectoire
}
return 0;
}

```

```
bool traj_indexation_show_error(int _error,tLabel *L){
```

```

if(_error == 0){
    LabelTextSet(L, "indexation terminee");
} else if(_error == 1){
    LabelTextSet(L, "erreur de lecture");
} else if(_error == 2){
    LabelTextSet(L, "trop de traj");
} else if(_error == 3){
    LabelTextSet(L, "erreur d ecriture");
} else if(_error == 4){
    LabelTextSet(L, "echec selection traj");
} else if(_error == 5){
    LabelTextSet(L, "apprentissage en cour");
}
return true;
}
```

```
bool traj_getNbTraj(tHandle _h,int *_nbTraj){
```

```

if(!LectureFNCT03R(_h,50,1)){return false;}
else {
    *_nbTraj = (int)lectureFNCT03[0];
    return true;
}
}
```

```
int traj_lancerT(int _indexT,tHandle *_h){  
    if(_indexT < 0 && _indexT > 32){return 4;}  
    ecritureFNCT06 = _indexT;  
    if(!EcritureFNCT06W(*_h,41)){return 3;} // selection traj. i sur le robot  
  
    int loopcount = 0;  
    do{  
        hmi_DelayUS(500);  
        LectureFNCT03R(*_h,51,1);  
        loopcount++;  
    } while(lectureFNCT03[0] == 0 && loopcount < 100);//selection de la trajectoire  
  
    ecritureFNCT06 = 0x1;  
    if(!EcritureFNCT06W(*_h,42)){return 1;}  
    else {  
  
        if(!EcritureFNCT06W(*_h,51)){return 2;}//desectionne la trajectoire  
    }  
    return 0;  
}  
  
bool traj_lancerT_show_error(int _error,tLabel *L){  
    switch(_error){  
        case 0:  
            LabelTextSet(L, "trajectoire Lancee");  
            break;  
        case 1:  
            LabelTextSet(L, "erreur de comm.");  
            break;  
        case 2:  
            LabelTextSet(L, "erreur de comm.");  
    }
```

```
        break;

    case 3:
        LabelTextSet(L, "erreur de comm.");
        break;

    case 4:
        LabelTextSet(L, "erreur traj trop bas haut");
        break;

    default:
        LabelTextSet(L, "erreur inconnue");
    }

    return true;
}

bool traj_arret(tHandle *_h, tLabel *L){
    bool err = false;

    ecritureFNCT06 = 0xffff;
    err = EcritureFNCT06W(*_h,43);

    hmi_DelayUS(100000);

    lectureFNCT03[0] = 0;
    LectureFNCT03R(*_h,43,1);
    if(err && lectureFNCT03[0] > 0){
        LabelTextSet(L,"signal envoyer");
    } else {
        LabelTextSet(L,"signal perdue");
    }

    return true;
}
```

```
#endif
```

Fin de niryo.h

## **Début de modbus.h**

```
#ifndef MODBUS_H
#define MODBUS_H

#define ArraySize 32

///////////////////////////////
//      variables de communication ModBus
///////////////////////////////

// variable pour les fonctions mtm fourni par HMIWorks
bool verrouMvmt = false;
int adressMemoire = 0;
int IDModBusServer = 1;
int nbDeMots = 1;
DWORD timeout = 200;
//mtm_ReadDO
char lectureFNCT01[ArraySize];
//mtm_ReadADI
char lectureFNCT02[ArraySize];
//mtm_ReadAO
WORD lectureFNCT03[ArraySize];
//mtm_ReadAI
WORD lectureFNCT04[ArraySize];
```

```
//mtm_WriteAO  
char ecritureFNCT05;  
  
//mtm_WriteAO  
WORD ecritureFNCT06;  
  
//mtm_WriteAO nbDeMots > 1  
char ecritureFNCT15[ArraySize];  
  
//mtm_WriteAO nbDeMots > 1  
WORD ecritureFNCT16[ArraySize];
```

```
///////////
```

```
//      fonctions de communication ModBus
```

```
///////////
```

```
bool LectureFNCT01R(tHandle h,int _adressMemoire,int _nbDeMots){  
    adressMemoire = _adressMemoire;//mise des variable dans les variables global  
    nbDeMots = _nbDeMots;  
    bool erreur = false;  
  
    memset(&lectureFNCT01, 0, ArraySize);//remet a zero la liste  
  
    erreur = mtm_ReadDO(h, IDModBusServer, adressMemoire, nbDeMots, lectureFNCT01,  
    timeout);//utilisation de la commande de la tablette  
  
    return erreur;// renvoie d'erreur  
}
```

```
bool LectureFNCT02R(tHandle h,int _adressMemoire,int _nbDeMots){  
    adressMemoire = _adressMemoire;  
    nbDeMots = _nbDeMots;  
    bool erreur = false;
```

```
        memset(&lectureFNCT02, 0, ArraySize);

        erreur = mtm_ReadDI(h, IDModBusServer, adressMemoire, nbDeMots, lectureFNCT02,
timeout);

        return erreur;
    }

bool LectureFNCT03R(tHandle h,int _adressMemoire,int _nbDeMots){

    adressMemoire = _adressMemoire;
    nbDeMots = _nbDeMots;
    bool erreur = false;

    for(int i = 0; i < ArraySize; i++){//remet a zero la liste
        lectureFNCT03[i] = 0;
    }

    erreur = mtm_ReadAO(h, IDModBusServer, adressMemoire, nbDeMots, lectureFNCT03,
timeout);

    return erreur;
}

bool LectureFNCT04R(tHandle h,int _adressMemoire,int _nbDeMots){

    adressMemoire = _adressMemoire;
    nbDeMots = _nbDeMots;
    bool erreur = false;

    for(int i = 0; i < ArraySize; i++){
```

```
lectureFNCT04[i] = 0;  
}  
  
erreur = mtm_ReadAI(h, IDModBusServer, adressMemoire, nbDeMots, lectureFNCT04,  
timeout);  
  
return erreur;  
}  
  
bool EcritureFNCT05W(tHandle h,int _adressMemoire){  
    adressMemoire = _adressMemoire;  
    nbDeMots = 1;  
    bool erreur = false;  
  
    erreur = mtm_WriteDO(h, IDModBusServer, adressMemoire, nbDeMots,  
&ecritureFNCT05, timeout);  
  
    //memset(&ecritureFNCT05, 0, 1);  
  
    return erreur;  
}  
  
bool EcritureFNCT06W(tHandle h,int _adressMemoire){  
    adressMemoire = _adressMemoire;  
    nbDeMots = 1;  
    bool erreur = false;  
  
    erreur = mtm_WriteAO(h, IDModBusServer, adressMemoire, nbDeMots,  
&ecritureFNCT06, timeout);
```

```
ecritureFNCT06 = 0;
```

```
return erreur;  
}
```

```
/*
```

afin de regler un problem avec la facon dont le robot communique en modbus et cesse de répondre lorsqu'un mouvement est lancer par la tablette cette fonction est une copie de EcritureFNCT06W mais avec un timeout de 10 000 ms

```
*/
```

```
bool EcritureFNCT06Wmvt(tHandle h,int _adressMemoire){
```

```
    adressMemoire = _adressMemoire;
```

```
    nbDeMots = 1;
```

```
    bool erreur = false;
```

```
    erreur = mtm_WriteAO(h, IDMModBusServer, adressMemoire, nbDeMots,  
&ecritureFNCT06, (DWORD)1500);
```

```
    ecritureFNCT06 = 0;
```

```
    return erreur;
```

```
}
```

//les fonctions suivante ne sont pas accepter par le niryo ned 2 et donc inutiliser

```
bool EcritureFNCT15W(tHandle h,int _adressMemoire,int _nbDeMots){
```

```
    adressMemoire = _adressMemoire;
```

```
nbDeMots = _nbDeMots;  
bool erreur = false;  
  
erreur = mtm_WriteDO(h, IDMModBusServer, adressMemoire, nbDeMots, ecritureFNCT15,  
timeout);  
  
memset(&ecritureFNCT15, 0, 1);  
  
return erreur;  
}  
  
bool EcritureFNCT16W(tHandle h,int _adressMemoire,int _nbDeMots){  
    adressMemoire = _adressMemoire;  
    nbDeMots = _nbDeMots;  
    bool erreur = false;  
  
    erreur = mtm_WriteAO(h, IDMModBusServer, adressMemoire, nbDeMots, ecritureFNCT16,  
timeout);  
  
    for(int i = 0; i < ArraySize; i++){  
        ecritureFNCT16[i] = 0;  
    }  
  
    return erreur;  
}
```

```
//////////  
// demande de mouvement ModBus  
//////////
```

```
int demandeDeMouvementNiryo(tHandle _h, int _adressMemoire,bool noExeT,bool  
noColT){//gere les verification pour empêcher les demandes lorsque le robot est en  
//mouvement(is executing command flag holding reg 150)  
  
    if(hmi_TCPState(_h) == 3){  
  
        LectureFNCT03R(_h,150,1);  
  
        LectureFNCT04R(_h,301,1);  
  
        if(lectureFNCT03[0] == 0 || noExeT){//si pas d'execution ou noExeT == true  
            if(lectureFNCT04[0] == 0 || noColT){//si pas de colision ou noColT == true  
                ecritureFNCT06 = 0xFFFF;  
  
                EcritureFNCT06Wmvmmt(_h,_adressMemoire);  
  
                hmi_DelayUS(500);  
  
            } else {  
  
                return 3;  
  
            }  
  
        } else {  
  
            return 2;  
  
        }  
  
    } else {  
  
        return 1;  
  
    }  
  
    return 0;  
}  
  
bool nettoyageColision(tHandle _h){  
  
    bool error = false;  
  
    ecritureFNCT06 = 0xFFFF;  
  
    error = EcritureFNCT06W(_h,103);  
  
    return error;  
}
```

```
///////////  
//          fin des fonctions de communication ModBus  
///////////
```

```
#endif
```

Fin de modbus.h

## Début de \_\_Connexion.h

```
tHandle h = -1;
```

```
tHandle h2 = -1;
```

```
tHandle hdebug = -1;
```

```
extern int serveurNormal;
```

```
extern int serveurCustom;
```

```
extern int connexionModBus(tHandle*,int);
```

```
/*
```

```
struct connection{
```

```
    tHandle srvrNormal;
```

```
    tHandle srvrCustom;
```

```
};
```

```
*/
```

```
int state = 0;
```

```
bool run = true;
```

```
int connexionStatus = 0;// 0 = pas dessai 1 = essaie en cour 2 = connecter
```

```
int connexionStatusSecondaire = 0;//
```

```
void Timer15OnExecute(tWidget *pWidget)
{
    connexionModBus(&h,serveurNormal);
    hmi_DelayUS(500000);
    connexionModBus(&h2,serveurCustom);
    /*
    hdebug = hmi_TCPNew();
    if(hdebug != -1){
        hmi_TCPOpen(hdebug,TCP_IPADDR(192,168,1,20), 12345, 42000);
        hmi_DelayUS(500);
        char t[32];
        sprintf(t,"test");
        hmi_TCPWrite(hdebug,(unsigned char*)t,32);
    }
    */
}
```

```
    TimerEnabledSet(&Timer29, 1);
    TimerEnabledSet(&Timer15, 0);
    hmi_GotoFrameByName("Statut");
}
```

```
void BitButton124OnClick(tWidget *pWidget)
{
    TimerEnabledSet(&Timer29, 1);
    TimerEnabledSet(&Timer15, 0);
}
```

```
    hmi_GotoFrameByName("Statut");  
}  
  
void Connexion12OnCreate()  
{  
    /*  
     unsigned char backlight[3];  
  
     hmi_UserParamsGet(0,1,backlight);  
     int ibacklight = ((int)backlight[0]);  
     if(ibacklight > 9 && ibacklight <256){  
         hmi_BacklightSet(backlight[0]);  
     }  
    */  
}
```

Fin de \_\_Connexion.h

## Début de \_\_Statut.h

```
/*  
  
//inclusions  
#include "conversion.h"  
#include "modbus.h"  
#include "connections.h"  
#include "niryo.h"  
  
extern tHandle h;  
extern tHandle h2;
```

```
extern bool trajectoireEnCour;

//variable du niryo ned 2
//table des axes en radiant
float axesRobot[6]={0.00,0.00,0.00,0.00,0.00,0.00};
//valeurs des axes cible en radiant
float axesRobotCible[6]={0.00,0.50,-1.25,0.00,0.00,0.00};
//serveurs
int serveurNormal = 1;
int serveurCustom = 2;

//outil equiper
WORD outilEquiper = 0;

tSliderWidget *SliderAxes[6];
tLabel *LabelAxes[6];
tPushButtonWidget *pushButtonAxes[12];

//si vraie toutes les axes sont affiché en degrées
extern bool deegree;

//error handling/gestion d'erreurs les fonction de hmi works renvoie vraie si l'action est réussi
bool errorCatching[6][10] = {false};
int disconnectedCount = 0;

///////////////////////////////
// Variables de textes car la fonction LabelTextSet n'a pas de buffer (et necesite un string par label)
/////////////////////////////
```

```
static char tabCharCible[6][32];//variables des label cible pour le robot  
static char tabCharAxes[6][32];//variables des slider des axes du robot  
static char tabCharAutre[10][32];//pour les autres labels
```

```
//////////
```

```
// fonction utile
```

```
//////////
```

```
void UpdateAllTargetValues(){  
    for(int i = 0; i<6 ; i++){  
        if(degree == true){  
            sprintf(tabCharCible[i], "%d", (int)(radianToDegree(axesRobotCible[i])));  
        } else {  
            FloatToStr(tabCharCible[i],axesRobotCible[i],3);  
        }  
        LabelTextSet(LabelAxes[i],tabCharCible[i]);  
    }  
}
```

```
//////////
```

```
// code principal (fonction bouttons, minuteur, etc)
```

```
//////////
```

```
bool led = false;
```

```
void Statut2OnCreate(){  
    SliderAxes[0] = &Slider9;  
    SliderAxes[1] = &Slider18;
```

```
SliderAxes[2] = &Slider19;  
SliderAxes[3] = &Slider24;  
SliderAxes[4] = &Slider25;  
SliderAxes[5] = &Slider26;
```

```
LabelAxes[0] = &Label34;  
LabelAxes[1] = &Label35;  
LabelAxes[2] = &Label36;  
LabelAxes[3] = &Label37;  
LabelAxes[4] = &Label38;  
LabelAxes[5] = &Label39;
```

```
pushButtonAxes[0] = &TextPushButton43;  
pushButtonAxes[1] = &TextPushButton44;  
pushButtonAxes[2] = &TextPushButton45;  
pushButtonAxes[3] = &TextPushButton46;  
pushButtonAxes[4] = &TextPushButton47;  
pushButtonAxes[5] = &TextPushButton48;  
pushButtonAxes[6] = &TextPushButton49;  
pushButtonAxes[7] = &TextPushButton50;  
pushButtonAxes[8] = &TextPushButton51;  
pushButtonAxes[9] = &TextPushButton52;  
pushButtonAxes[10] = &TextPushButton53;  
pushButtonAxes[11] = &TextPushButton54;
```

```
for(int i = 0; i < 6; i++){  
    sprintf(tabCharCible[i], "%d" , (int)(axesRobotCible[i]*1000.00));  
    LabelTextSet(LabelAxes[i],tabCharCible[i]);  
}
```

```
UpdateAllTargetValues();
```

```
}
```

```
void TextPushButton13OnClick(tWidget *pWidget){ //boutton activer / attraper objet  
    mvmt_toggleOutil(&h,&Label28,true,outilEquiper);  
}
```

```
void TextPushButton6OnClick(tWidget *pWidget){ //boutton desactiver outil / lacher objet  
    mvmt_toggleOutil(&h,&Label28,false,outilEquiper);  
}
```

```
void Timer8OnExecute(tWidget *pWidget){      //minuteur pour le code a executer une fois par  
seconde
```

```
    static char strDebug3000[32];  
    usnprintf(strDebug3000, sizeof(strDebug3000), "h : %X %X", h, &h );  
    LabelTextSet(&Label212,strDebug3000);  
    static char strDebug3001[32];  
    usnprintf(strDebug3001, sizeof(strDebug3001), "h2 : %X %X", h2, &h2);  
    LabelTextSet(&Label213,strDebug3001);
```

```
    UpdateAllTargetValues();
```

```
    LectureFNCT04R(h,402,2);
```

```
    bool calibration = (lectureFNCT04[0] == 0);
```

```

SliderValueSet(&Slider11, lectureFNCT04[1]);
usnprintf(tabCharAutre[0], sizeof(tabCharAutre[0]), "%d", lectureFNCT04[1]);
SliderTextSet(&Slider11,tabCharAutre[1]);

errorCatching[0][0] = LectureFNCT04R(h,0,6); // lecture des axes ----

if(errorCatching[0][0] == false){
    LabelTextSet(&Label41, "connexion perdue");
    for(int i = 0; i < 6 ; i++){
        SliderTextSet(SliderAxes[i], "deconnecter");
    }
} else {
    LabelTextSet(&Label41, "connexion OK");
    for(int i = 0; i<6;i++){
        axesRobot[i] = axesToRadian(lectureFNCT04[i]);
    }
}

//char axeName[6][11]={ "J1-Base","J2-Shoulder","J3-Elbow","J4-Forearm","J5-
Wrist","J6-Hand"};

for(int i = 0; i < 6 ; i++){
    SliderValueSet(SliderAxes[i], (int)(axesRobot[i]*1000));
    if(degree == true){
        //snprintf(tabCharAxes[i], sizeof(tabCharAxes[i]), "%s
%d",axeName[i], (int)(radianToDegree(axesRobot[i])));
        snprintf(tabCharAxes[i], sizeof(tabCharAxes[i]), "J%d %d",i+1, (int)
(radianToDegree(axesRobot[i])));
        //usnprintf(tabCharAxes[i], sizeof(tabCharAxes[i]), "%d", (int)
(radianToDegree(axesRobot[i])));
    } else {
}
}

```

```

        char tstr[32];
        FloatToStr(tstr,axesRobot[i],3);
        //usnprintf(tabCharAxes[i], sizeof(tabCharAxes[i]), "J %d", (int)
(axesRobot[i]*1000));
        //usnprintf(tabCharAxes[i], sizeof(tabCharAxes[i]), "%s
%s",axeName[i], tstr );
        usnprintf(tabCharAxes[i], sizeof(tabCharAxes[i]), "J%d %s",i+1,
tstr );
    }
    SliderTextSet(SliderAxes[i], tabCharAxes[i]);
}

```

{}

```

//bool error200 = false;
LectureFNCT03R(h,150,3);//lecture des adresse pour les resultats de commande

```

```
    LectureFNCT04R(h,301,1);
```

```
    if(true){
```

```
        /*

```

```
        enum codeDErreur{
```

```
    };
```

```
    */

```

```
if(lectureFNCT03[1] == 0){
```

```
    sprintf(tabCharAutre[1],"attente de resultat");
```

```
} else if(lectureFNCT03[1] == 1){
```

```
    //LabelTextSet(&Label5, "success");
```

```
    sprintf(tabCharAutre[1],"success");
```

```
} else if(lectureFNCT03[1] == 2){
```

```

sprintf(tabCharAutre[1],"commande rejeter");//parametre non valid etc
} else if(lectureFNCT03[1] == 3){

    sprintf(tabCharAutre[1],"commande abandoner");

} else if(lectureFNCT03[1] == 4){

    sprintf(tabCharAutre[1],"commande annuler");

} else if(lectureFNCT03[1] == 5){

    sprintf(tabCharAutre[1],"erreure inattendue");

} else if(lectureFNCT03[1] == 6){

    sprintf(tabCharAutre[1],"delai depasser");

} else if(lectureFNCT03[1] == 7){

    sprintf(tabCharAutre[1],"erreure interne");

} else if(lectureFNCT03[1] == 8){

    sprintf(tabCharAutre[1],"commande deja en cour");

} else if(lectureFNCT03[1] == 9){

    sprintf(tabCharAutre[1],"colision");

}

} else {

//LabelTextSet(&Label5, "valeur inattendue");

    sprintf(tabCharAutre[1],"valeur inattendue");

//usnprintf(tabCharAutre[1], sizeof(tabCharAutre[1]), "%X",
lectureFNCT04[0]);

//LabelTextSet(&Label5,tabCharAutre[1]);

}

usprintf(tabCharAutre[2], "%s : %X",tabCharAutre[1], lectureFNCT03[2]);

LabelTextSet(&Label5,tabCharAutre[2]);

if(lectureFNCT04[0] == 0){//affichage label 66

    if(lectureFNCT03[0] == 1){

        if(calibration){

}

```

```
    LabelTextSet(&Label66, "calibration en cour");
} else {
    LabelTextSet(&Label66, "execute cmd");
}
} else {
    LabelTextSet(&Label66, "pas d execution");
}
} else {
    LabelTextSet(&Label66, "colision detecte");
}

if(degree == true){
    LabelTextSet(&Label58,"degree");
} else {
    LabelTextSet(&Label58,"radian");
}
}
```

errorCatching[0][2] = LectureFNCT04R(h,200,1); //lecture de l'outil -----

```
if(errorCatching[0][2] == true){

static char stroutil[2][32];
memset(stroutil[0],0,sizeof(stroutil));
sprintf(stroutil[0],"outil: ");

outilEquiper = lectureFNCT04[0];

if(lectureFNCT04[0] == 31){
```

```
        strcat(stroutil[0], "pompe a vide" );
    } else if(lectureFNCT04[0] == 11){
        strcat(stroutil[0], "pince 1" );
    } else if(lectureFNCT04[0] == 12){
        strcat(stroutil[0], "pince 2" );
    } else if(lectureFNCT04[0] == 13){
        strcat(stroutil[0], "pince 3" );
    } else {
        strcat(stroutil[0], "inconnu" );
    }
LabelTextSet(&Label21, stroutil[0]);

// if(lectureFNCT04[0] == 31){
//     sprintf(stroutil[1],"pompe");
// } else if(lectureFNCT04[0] == 11 || lectureFNCT04[0] == 12 || lectureFNCT04[0]
== 13){
    // sprintf(stroutil[1],"pince");
    // } else {
    //     sprintf(stroutil[1],"inconnue");
    // }
// LabelTextSet(&Label31, stroutil[1]);
}

LectureFNCT03R(h,100,1);
usnprintf(tabCharAutre[9], sizeof(tabCharAutre[9]), "%X", lectureFNCT03[0]);
LabelTextSet(&Label201,tabCharAutre[9]);
}
```

```
void Timer10OnExecute(tWidget *pWidget){ //test indicateur led
```

```
    if(led == true){  
        hmi_SetLED(HMI_LED_ON);  
        led = false;  
    } else {  
        hmi_SetLED(HMI_LED_OFF);  
        led = true;  
    }  
}
```

```
void Slider11OnSliderChange(tWidget *pWidget, long lValue){ //slider de test(demande la  
température du robot niryo)
```

```
    static char strValue[10];  
    usnprintf(strValue , sizeof(strValue) , "%d" , lValue);  
    SliderTextSet(&Slider11 , strValue);  
    WidgetPaint((tWidget*)&Slider11);  
}
```

```
void BitButton22OnClick(tWidget *pWidget){ //boutton demarrer mouvement
```

```
    mvmt_niryo_show_error(mvmt_niryo(axesRobotCible,&h,serveurNormal,false),&Label28);  
}
```

```
void Timer29OnExecute(tWidget *pWidget){ //detection de deconnexion
```

```
    //int erreurs[2] = {0,0};  
    //static char temp[2][32];  
    checkConnexion(&h,serveurNormal);  
    checkConnexion(&h2,serveurCustom);  
    //usnprintf(temp[0], sizeof(temp[0]), "erreur : %d", erreurs[0]);  
    //LabelTextSet(&Label5,temp[0]);  
    //usnprintf(temp[1], sizeof(temp[1]), "erreur : %d", erreurs[1]);  
    //LabelTextSet(&Label20,temp[1]);  
}
```

```
void TextPushButton32OnClick(tWidget *pWidget){ //remise a la position par defaut
```

```
    float axesRobotTemp[6]={0.00,0.50,-1.25,0.00,0.00,0.00};  
  
    mvmt_niryo_show_error(mvmt_niryo(axesRobotTemp,&h,serveurNormal,false),&Label28);  
}
```

```
void TextPushButton193OnClick(tWidget *pWidget)
```

```
{  
    float axesRobotTemp[6]={0.00,0.00,0.00,0.00,0.00,0.00};  
  
    mvmt_niryo_show_error(mvmt_niryo(axesRobotTemp,&h,serveurNormal,false),&Label28);
```

```
}
```

```
void TextPushButton33OnClick(tWidget *pWidget)
{
    for(int i = 0; i<6 ; i++ ){
        axesRobotCible[i] = axesRobot[i];
    }
    UpdateAllTargetValues();
}
```

```
void SliderAxeOnChange(tWidget *pWidget, long lValue){ // mise a jour de la valeur cible quand
un axe est changer
```

```
for(int i = 0; i < 6 ; i++){
    if(pWidget != (tWidget*)SliderAxes[i]){
        continue;
    } else {
        if(degree == true){
            usnprintf(tabCharCible[i] , sizeof(tabCharCible[i]) , "%d" , (int)
(radianToDegree(((float)lValue)/1000.00)));//degree
        } else {
            FloatToStr(tabCharCible[i],((float)lValue)/1000),3); //radian
            //usnprintf(tabCharCible[i] , sizeof(tabCharCible[i]) , "%d" ,
(int)lValue);
        }
        SliderTextSet(SliderAxes[i] , tabCharCible[i]);
        WidgetPaint((tWidget*)SliderAxes[i]);
        axesRobotCible[i] = (((float)lValue)/1000);
        LabelTextSet(LabelAxes[i],tabCharCible[i]);
    }
}
```

```
    }  
}  
}  
  
void TextButtonOnClick(tWidget *pWidget){  
    for(int i = 0; i < 12 ; i++){  
        if(pWidget != (tWidget*)pushButtonAxes[i]){  
            continue;  
        } else {  
            if(i < 6){  
                long min,max;  
                SliderRangeGet(SliderAxes[i],min,max);  
                if(min == 0){min = 0;}//evite le warning du compilateur  
                float fmax = (((float)max)/1000);  
                if(axesRobotCible[i] < fmax){  
                    axesRobotCible[i] = axesRobotCible[i] + 0.010;  
                }  
                if(axesRobotCible[i] > fmax){  
                    axesRobotCible[i] = fmax;  
                }  
                if(degree == true){  
                    sprintf(tabCharCible[i], "%d" , (int)  
(radianToDegree(axesRobotCible[i])));//degree  
                } else {  
                    FloatToStr(tabCharCible[i],axesRobotCible[i],3);//radian
```

```
//sprintf(tabCharCible[i], "%d" , (int)
(axesRobotCible[i]*1000.00));
}

LabelTextSet(LabelAxes[i],tabCharCible[i]);

} else if (i >= 6 && i < 12){

    long min,max;
    SliderRangeGet(SliderAxes[i-6],min,max);
    if(max == 0){max = 0;}//evite le warning du compilateur
    float fmin = (((float)min)/1000);

    if(axesRobotCible[i-6] > fmin){
        axesRobotCible[i-6] = axesRobotCible[i-6] - 0.010;
    }
    if(axesRobotCible[i-6] < fmin){
        axesRobotCible[i-6] = fmin;
    }
    if(degree == true){
        sprintf(tabCharCible[i-6], "%d" , (int)
(radianToDegree(axesRobotCible[i-6])));
    } else {
        FloatToStr(tabCharCible[i],axesRobotCible[i-6],3);
        //sprintf(tabCharCible[i-6], "%d" , (int)(axesRobotCible[i-
6]*1000.00));
    }
    LabelTextSet(LabelAxes[i-6],tabCharCible[i-6]);
}

}

}
```

```
void BitButton62OnClick(tWidget *pWidget){  
}  
  
void TextPushButton147OnClick(tWidget *pWidget){ //arret  
    mvmt_arret(&h,&Label41);  
}  
  
Fin de __Statut.h
```

## Début de \_\_Trajectoire.h

```
tPushButtonWidget *pbTrajectoire[10];  
int selectedBoutton = -1;  
int selectedTrajectory = 1;  
  
extern tHandle h;  
extern tHandle h2;  
bool trajectoirEnCour = false;  
//recuperation des fonctions de modbus.h  
extern char lectureFNCT01[];  
extern bool LectureFNCT01R(tHandle,int,int);  
extern char lectureFNCT02[];  
extern bool LectureFNCT02R(tHandle,int,int);  
extern WORD lectureFNCT03[];  
extern bool LectureFNCT03R(tHandle,int,int);  
extern WORD lectureFNCT04[];
```

```
extern bool LectureFNCT04R(tHandle,int,int);  
extern char lectureFNCT05;  
extern bool EcritureFNCT05W(tHandle,int);  
extern WORD ecritureFNCT06;  
extern bool EcritureFNCT06W(tHandle,int);
```

```
extern int serveurNormal;  
extern int serveurCustom;  
extern int checkConnexion(tHandle*,int);
```

```
extern bool mvmt_arret(tHandle*,tLabel*);
```

```
extern bool traj_arret(tHandle*,tLabel*);
```

```
typedef struct {
```

```
    int ID;  
    char name[21];  
    int index;
```

```
} trajectory ;
```

```
extern int traj_indexation(trajetory t[],int,tHandle);  
extern bool traj_indexation_show_error(int,tLabel*);  
extern bool traj_getNbTraj(tHandle,int*);  
extern int traj_lancerT(int,tHandle*);
```

```
int nombreDeTrajectoires = 0;
```

```
static char trajectoires[32][32];
```

```
int offset = 0;
```

```
trajetory traj[32];
```

```
void Trajectoire27OnCreate(){

    pbTrajectoire[0] = &TextPushButton111;// flech haut
    pbTrajectoire[1] = &TextPushButton117;
    pbTrajectoire[2] = &TextPushButton122;
    pbTrajectoire[3] = &TextPushButton123;
    pbTrajectoire[4] = &TextPushButton125;
    pbTrajectoire[5] = &TextPushButton130;
    pbTrajectoire[6] = &TextPushButton131;
    pbTrajectoire[7] = &TextPushButton132;
    pbTrajectoire[8] = &TextPushButton133;
    pbTrajectoire[9] = &TextPushButton134;// flech bas

    for(int i = 0 ; i < 32 ; i++){
        sprintf(trajectoires[i],"vide %d",i);
    }

    for(int i = 1 ; i < 9 ; i++){
        TextButtonTextSet(pbTrajectoire[i],trajectoires[i+(offset-1)]);
    }

}

static char str1[32];

void TextPushButtonTrajectorieSelectionOnRelease(tWidget *pWidget){

    for(int i = 0; i < 10 ; i++){
        if(pWidget != (tWidget*)pbTrajectoire[i]){
            continue;
        } else {
            selectedBoutton = i;
        }
    }
}
```

```
for(int i = 0; i < 10 ;i++){  
    PushButtonFillColorSet(pbTrajectoire[i],0x646464);  
    WidgetPaint((tWidget*)pbTrajectoire[i]);//redessine les boutton pour changer la  
couleur  
}  
  
if(selectedBoutton >= 0){  
    PushButtonFillColorSet(pbTrajectoire[selectedBoutton],0x00ff40);  
    //TextButtonTextSet(pbTrajectoire[selectedBoutton],str1);  
}  
  
sprintf(str1,"%d",selectedBoutton);  
}  
  
void Timer135OnExecute(tWidget *pWidget){  
    static char strDebug3000[32];  
    usnprintf(strDebug3000, sizeof(strDebug3000), "h : %X", h);  
    LabelTextSet(&Label210,strDebug3000);  
    static char strDebug3001[32];  
    usnprintf(strDebug3001, sizeof(strDebug3001), "h2 : %X", h2);  
    LabelTextSet(&Label211,strDebug3001);  
  
    bool arrowPressed = false;  
  
    if(selectedBoutton == 9 || selectedBoutton == 0){  
        arrowPressed = true;  
    }  
  
    if(selectedBoutton == 9 && offset < 23){
```

```
offset = offset + 2;

} else if ((selectedBoutton == 9 && offset == 23)){

    offset++;

} else if (selectedBoutton == 0 && offset > 1){

    offset = offset - 2;

} else if ((selectedBoutton == 0 && offset == 1)){

    offset--;

} else {

}

if(arrowPressed){

    selectedBoutton = -1;

    for(int i = 1 ; i < 9 ; i++){

        TextButtonTextSet(pbTrajectoire[i],trajectoires[i+(offset-1)]);

    }

    arrowPressed = false;

    for(int i = 0; i < 10 ;i++){

        PushButtonFillColorSet(pbTrajectoire[i],0x646464);

        WidgetPaint((tWidget*)pbTrajectoire[i]);

    }

    if((selectedTrajectory - offset) > 0 && selectedTrajectory - offset < 9){

        PushButtonFillColorSet(pbTrajectoire[(selectedTrajectory - offset)],0x00ff40);

    }

}

static char txt1[32];

if(selectedBoutton >= 0 ){

    selectedTrajectory = (selectedBoutton+offset);

}

sprintf(txt1,"%d",(selectedTrajectory));
```

```
LabelTextSet(&Label208,txt1);

}

void Timer139OnExecute(tWidget *pWidget){ // verification periodique de la connexion au niryo

    checkConnexion(&h,serveurNormal);
    checkConnexion(&h2,serveurCustom);
}

int LancerTrajectoire(int _trajectoire){

    if(selectedTrajectory > 0 && selectedTrajectory <= 32){

        traj_lancerT(_trajectoire-1,&h2);
        trajectoireEnCour = true;
    }

    return 0;
}

void TextPushButton144OnRelease(tWidget *pWidget){ // boutton lancer trajectoire

    LancerTrajectoire(selectedBoutton);
}

void TextPushButton42OnClick(tWidget *pWidget){ //arret

    mvmt_arret(&h,&Label151);
    traj_arret(&h2,&Label151);
}

void TextPushButton144OnClick(tWidget *pWidget){ //lance une trajectoire
```

```
}
```

```
void Timer206OnExecute(tWidget *pWidget){  
    if(trajecoirEnCour){return;}  
    traj_indexation_show_error(traj_indexation(traj,32,h2),&Label151);  
  
    traj_getNbTraj(h2,&nombreDeTrajectoires);  
  
    static char txt2[32];  
    sprintf(txt2,"%d",(nombreDeTrajectoires));  
    LabelTextSet(&Label209,txt2);  
  
    if(nombreDeTrajectoires < 0){nombreDeTrajectoires = 0;}  
    if(nombreDeTrajectoires > 32){nombreDeTrajectoires = 32;}  
  
    int x = 0;  
    while(x < nombreDeTrajectoires){  
  
        sprintf(trajecoirs[x],"%s - %d",traj[x].name,traj[x].ID);  
        //sprintf(trajecoirs[i],"test ligne %d",i+10);  
        x++;  
    }  
  
    for(int i = 1 ; i < 9 ; i++){  
        TextButtonTextSet(pbTrajectoire[i],trajecoirs[i+(offset-1)]);  
    }  
}
```

```
void TextPushButton57OnClick(tWidget *pWidget)
{
}
```

Fin de \_\_Trajectoire.h

## Début de \_\_Statistic.h

```
extern tHandle h;
extern tHandle h2;
extern char lectureFNCT01[];
extern bool LectureFNCT01R(tHandle,int,int);
extern char lectureFNCT02[];
extern bool LectureFNCT02R(tHandle,int,int);
extern WORD lectureFNCT03[];
extern bool LectureFNCT03R(tHandle,int,int);
extern WORD lectureFNCT04[];
extern bool LectureFNCT04R(tHandle,int,int);
extern char lectureFNCT05;
extern bool EcritureFNCT05W(tHandle,int);
extern WORD ecritureFNCT06;
extern bool EcritureFNCT06W(tHandle,int);

extern int serveurNormal;
extern int serveurCustom;
extern int checkConnexion(tHandle*,int);

extern bool mvmt_arret(tHandle*,tLabel*);
```

```
static char tabCharStat[10][32];

void Timer115OnExecute(tWidget *pWidget){

//LectureFNCT04R(h2,1000,1);//lecture des adresses de test

    static char test[32];

    sprintf(test, "%d" , (int)lectureFNCT04[0]);

    LabelTextSet(&Label114,test);

    bool works = false;//si il y a une erreur la variable va rester a false

    works = LectureFNCT04R(h,400,10);

    if(works == true){

        if(lectureFNCT04[0] == 1){LabelTextSet(&Label140,"moteur OK");
        } else if(lectureFNCT04[0] == 0){LabelTextSet(&Label140,"moteurs pas OK");
        } else {LabelTextSet(&Label140,"valeur non reconnu");}

        if(lectureFNCT04[1] == 1){LabelTextSet(&Label141,"calibration necessaire");
        } else if(lectureFNCT04[1] == 0){LabelTextSet(&Label141,"calibration non
necessaire");
        } else {LabelTextSet(&Label141,"valeur non reconnu");}

        if(lectureFNCT04[2] == 1){LabelTextSet(&Label142,"calibration en cour");
        } else if(lectureFNCT04[2] == 0){LabelTextSet(&Label142,"pas de calibration en
cour");
        } else {LabelTextSet(&Label142,"valeur non reconnu");}
```

```
SliderValueSet(&Slider137, lectureFNCT04[3]);  
usnprintf(tabCharStat[3], sizeof(tabCharStat[3]), "temperature cpu : %d",  
lectureFNCT04[3]);  
SliderTextSet(&Slider137,tabCharStat[3]);  
  
usnprintf(tabCharStat[9], sizeof(tabCharStat[9]), "HWversion : %X",  
lectureFNCT04[9]);  
LabelTextSet(&Label143,tabCharStat[9]);  
}  
works = false;  
works = LectureFNCT04R(h,301,1);  
  
if(works == true){  
    if(lectureFNCT04[0] == 1){LabelTextSet(&Label153,"colision detecte");  
    } else if(lectureFNCT04[0] == 0){LabelTextSet(&Label153,"pas de colision");  
    }  
}  
}  
  
void Timer146OnExecute(tWidget *pWidget){  
    checkConnexion(&h,serveurNormal);  
    checkConnexion(&h2,serveurCustom);  
}  
  
void TextPushButton149OnClick(tWidget *pWidget){ //arret  
    mvmt_arret(&h,&Label150);  
}  
  
Fin de __Statistic.h
```

## Début de \_\_StatisticMoteurs.h

```
extern tHandle h;  
extern tHandle h2;  
  
extern WORD lectureFNCT03[];  
extern bool LectureFNCT03R(tHandle,int,int);  
extern WORD lectureFNCT04[];  
extern bool LectureFNCT04R(tHandle,int,int);  
extern char lectureFNCT05;  
extern bool EcritureFNCT05W(tHandle,int);  
extern WORD ecritureFNCT06;  
extern bool EcritureFNCT06W(tHandle,int);  
  
extern int serveurNormal;  
extern int serveurCustom;  
extern int checkConnexion(tHandle*,int);  
  
extern int customDataToInt(WORD);  
extern float customDataToFloat(WORD);  
  
tLabel *LabelVoltage[8];  
tLabel *LabelTemperature[8];  
tLabel *LabelCodeErreur[8];  
  
static char tabCharVolt[8][32];  
static char tabCharTemp[8][32];  
static char tabCharErreur[8][32];
```

```
void StatisticMoteurs17OnCreate()
{
    LabelVoltage[0] = &Label160;
    LabelVoltage[1] = &Label161;
    LabelVoltage[2] = &Label165;
    LabelVoltage[3] = &Label168;
    LabelVoltage[4] = &Label171;
    LabelVoltage[5] = &Label174;
    LabelVoltage[6] = &Label177;
    LabelVoltage[7] = &Label180;

    LabelTemperature[0] = &Label162;
    LabelTemperature[1] = &Label163;
    LabelTemperature[2] = &Label166;
    LabelTemperature[3] = &Label169;
    LabelTemperature[4] = &Label172;
    LabelTemperature[5] = &Label175;
    LabelTemperature[6] = &Label178;
    LabelTemperature[7] = &Label181;

    LabelCodeErreur[0] = &Label184;
    LabelCodeErreur[1] = &Label185;
    LabelCodeErreur[2] = &Label186;
    LabelCodeErreur[3] = &Label187;
    LabelCodeErreur[4] = &Label188;
    LabelCodeErreur[5] = &Label189;
    LabelCodeErreur[6] = &Label190;
    LabelCodeErreur[7] = &Label191;

}
```

```
void Timer183OnExecute(tWidget *pWidget)
{
    checkConnexion(&h,serveurNormal);
    checkConnexion(&h2,serveurCustom);
}

void Timer182OnExecute(tWidget *pWidget)
{
    bool works = false;
    works = LectureFNCT04R(h,410,24);
    if(works){
        for(int i = 0;i<8;i++){
            char temp[32];
            FloatToStr(temp,customDataToFloat(lectureFNCT04[i+8]),3);
            usnprintf(tabCharVolt[i], sizeof(tabCharVolt[i]), "%s V", temp);
            LabelTextSet(LabelVoltage[i],tabCharVolt[i]);
            usnprintf(tabCharTemp[i], sizeof(tabCharTemp[i]), "%d C",
((int)lectureFNCT04[i]));
            LabelTextSet(LabelTemperature[i],tabCharTemp[i]);
            usnprintf(tabCharErreur[i], sizeof(tabCharErreur[i]), "%d",
((int)lectureFNCT04[i+16]));
            LabelTextSet(LabelCodeErreur[i],tabCharErreur[i]);
        }
    }
}
```

Fin de \_\_StatisticMoteurs.h

## Début de \_\_Menu.h

```
extern tHandle h;
extern tHandle h2;
extern WORD lectureFNCT04[];
extern bool LectureFNCT04R(tHandle,int,int);
extern WORD ecritureFNCT06;
extern bool EcritureFNCT06W(tHandle,int);
extern bool EcritureFNCT06Wmvmt(tHandle,int);

bool degree = true;

extern int serveurNormal;
extern int serveurCustom;
extern int checkConnexion(tHandle*,int);

extern bool nettoyageColision(tHandle);
extern bool mvmt_arret(tHandle*,tLabel*);
extern int mvmt_calibration(tHandle*,int,bool);
extern bool mvmt_calibration_show_error(int,tLabel*);
extern bool mvmt_detect_tool(tHandle*,tLabel*);

void Menu61OnCreate(){
    if(degree == true){
```

```
    TextButtonTextSet(&TextPushButton64,"unite : degree");
} else {
    TextButtonTextSet(&TextPushButton64,"unite : radiant");
}

LabelTextSet(&Label112, "message");

int i = ((int)hmi_BacklightGet());
SliderValueSet(&Slider40,i);
}

void TextPushButton64OnRelease(tWidget *pWidget){
    degree = !degree;
    if(degree == true){
        TextButtonTextSet(&TextPushButton64,"unite : degree");
    } else {
        TextButtonTextSet(&TextPushButton64,"unite : radiant");
    }
}

void TextPushButton65OnRelease(tWidget *pWidget){ //calibration auto
    mvmt_calibration_show_error(mvmt_calibration(&h,serveurNormal,true),&Label112);
}

void TextPushButton30OnRelease(tWidget *pWidget){
    mvmt_detect_tool(&h,&Label112);
}

void Timer145OnExecute(tWidget *pWidget){
    checkConnexion(&h,serveurNormal);
    checkConnexion(&h2,serveurCustom);
}
```

```
void TextPushButton148OnClick(tWidget *pWidget)//arret
{
    mvmt_arret(&h,&Label112);
}
```

```
void TextPushButton152OnRelease(tWidget *pWidget){

    if(nettoyageColision(h))//
    {
        LabelTextSet(&Label112, "colision nettoyer");
    } else {
        LabelTextSet(&Label112, "nettoyage echoue");
    }
}
```

```
void Slider40OnSliderChange(tWidget *pWidget, long lValue){
    if(lValue < 256 && lValue > 0){
        unsigned char b = (unsigned char)lValue;
        hmi_BacklightSet(b);
        //hmi_UserParamsSet(0,1,b);
    }
}
```

```
void Slider200OnSliderChange(tWidget *pWidget, long lValue)
{
    static char str200[32];
    usnprintf(str200 , sizeof(str200) , "%d" , (int)lValue);
    ecritureFNCT06 = (WORD)lValue;
```

```
EcritureFNCT06W(h,514);  
SliderTextSet(&Slider200, str200);  
}  
  
Fin de __Menu.h
```

## Début de \_\_Clavier.h

```
//frame pour le clavier  
  
tPushButtonWidget *pb[39];  
  
#define PB_COUNT() (sizeof(pb)/sizeof(pb[0]))  
  
extern tHandle h;  
extern tHandle h2;  
  
extern int serveurNormal;  
extern int serveurCustom;  
extern int checkConnexion(tHandle,int);  
  
  
char charList[36] =  
{'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','  
W','X','Y','Z'};  
  
void Clavier23OnCreate()  
{  
    pb[0] = &TextPushButton55; // 0
```

```
pb[1] = &TextPushButton67; // 1
pb[2] = &TextPushButton68; // 2
pb[3] = &TextPushButton69; // 3
pb[4] = &TextPushButton70; // 4
pb[5] = &TextPushButton71; // 5
pb[6] = &TextPushButton72; // 6
pb[7] = &TextPushButton73; // 7
pb[8] = &TextPushButton74; // 8
pb[9] = &TextPushButton75; // 9
pb[10] = &TextPushButton76; // A
pb[11] = &TextPushButton77; // B
pb[12] = &TextPushButton78; // C
pb[13] = &TextPushButton79; // D
pb[14] = &TextPushButton80; // E
pb[15] = &TextPushButton81; // F
pb[16] = &TextPushButton82; // G
pb[17] = &TextPushButton83; // H
pb[18] = &TextPushButton84; // I
pb[19] = &TextPushButton85; // J
pb[20] = &TextPushButton86; // K
pb[21] = &TextPushButton87; // L
pb[22] = &TextPushButton88; // M
pb[23] = &TextPushButton89; // N
pb[24] = &TextPushButton90; // O
pb[25] = &TextPushButton91; // P
pb[26] = &TextPushButton92; // Q
pb[27] = &TextPushButton93; // R
pb[28] = &TextPushButton102; // S 102
pb[29] = &TextPushButton94; // T
pb[30] = &TextPushButton95; // U
pb[31] = &TextPushButton96; // V
```

```
pb[32] = &TextPushButton97; // W
pb[33] = &TextPushButton98; // X
pb[34] = &TextPushButton99; // Y
pb[35] = &TextPushButton100; // Z
pb[36] = &TextPushButton100; // effacer
pb[37] = &TextPushButton100; // annuler
pb[38] = &TextPushButton100; // entrer

}
```

```
//static char temp[100];
//static char starsign[15];
//static int flag;

static char outString[40];

void TextPushButtonClavierOnClick(tWidget *pWidget)
{
    static int positionCurseur = 0;
    char str[10];

    for(int i = 0 ; i < PB_COUNT() ; i++)
    {
        if(pWidget != (tWidget*)pb[i]) continue; //si widget different de pushbutton[i]
        continue

        if(i<36){
            usnprintf(str, sizeof(str), "%c", charList[i]);// char list vers char[]
        }
    }
}
```

```
    strcat(outString , str) ;//temp append str  
    //LabelTextSet(&Label65, outString); //afficher  
} else if(i==36){ //effacer  
    positionCurseur--;  
    if(positionCurseur < 0) positionCurseur = 0;  
    outString[positionCurseur] = '\0';  
} else if(i==37){ //annuler  
  
} else if(i==38){ //entree  
    hmi_GotoFrameByName("Trajectoire");  
}  
  
}  
}
```

Fin de \_\_Clavier.h