

III - LE PROGRAMME SUR ARDUINO DUE

Ce document se trouve sur ce site : http://www.la-tour.info/uts/uts_page15.html#conduite

Si vous réaliser des améliorations matériels ou logicielles, merci de les poster sur le forum RMF pour en faire profiter le plus grand nombre.

Si vous publier cette réalisation avec ou sans améliorations, vous devez publier votre code source.

Ce logiciel est un logiciel libre. Exigence du concepteur : Ne pas modifier la ligne d'affichage "JLF xx/xx/xxxx" sur l'écran LCD.

On peut modifier ce programme et le diffuser. Dans ce cas, il faut préciser l'origine et donner accès aux sources modifiées.

Définition : Un logiciel libre est un logiciel distribué avec l'intégralité de ses programmes-sources, afin que l'ensemble des utilisateurs qui l'emploient, puissent l'enrichir et le redistribuer à leur tour.

Note : Un logiciel libre n'est pas nécessairement gratuit et les droits de la chaîne des auteurs sont préservés.

Équivalent étranger : free software, open source software.

(Source : Vocabulaire de l'informatique (liste de termes, expressions et définitions adoptés), NOR: CTNX0710138K, J.O n° 93 du 20 avril 2007 page 7078, texte n° 84)

logiciel libre

Par logiciel libre on entend un logiciel qui offre la liberté aux utilisateurs d'exécuter, de copier, de distribuer, d'étudier, de modifier et d'améliorer le logiciel. Plus précisément, elle fait référence à quatre types de liberté pour les utilisateurs du logiciel :

La liberté d'exécuter le programme, pour tous les usages (liberté 0).

La liberté d'étudier comment le programme fonctionne et de l'adapter à ses besoins (liberté 1). L'accès au code source est une condition requise.

La liberté de redistribuer des copies, (liberté 2).

La liberté d'améliorer le programme et de diffuser les améliorations au public pour en faire profiter toute la communauté (liberté 3). L'accès au code source est une condition requise.

Généralités

Ce montage peut être adapté à n'importe quel simulateur sur ordinateur, du moment que l'Arduino puisse recevoir et envoyer des informations à l'ordinateur.

La plaque de montage de l'Arduino DUE est compatible.

Le programme fonctionne avec le logiciel "Train Simulator Classic RailWorks de DTG", mais il est adaptable à d'autres environnements.

Les périphériques I2C fonctionnent à une fréquence standard de 100 kHz.

Si on ajoute plus d'entrées/sorties, ou si l'on veut aller plus vite, on peut essayer de passer en mode Fast à 400 kHz.

La connexion série à l'ordinateur est à **38400** bps. Si l'on constate des problèmes on peut passer à 19200 ou 9600 bps.

Mon logiciel se compose de trois éléments :

- | | |
|--------------------------------|--|
| • PDC_Arduino_JLF.ino | Programme principal. Il est compatible avec toutes les configurations. |
| • Data_Recues_du_PC_TSCRJI.ino | Programme de réception des données. |
| • Data_Recues_du_PC_TSCRJI.ino | Programme d'envoi des données. |

Le code existe en plusieurs version, adapté à chaque locomotive. Pour le moment, c'est décliné pour deux locomotives.

- Les BB7200 de SimExpress, répertoire : **PDC_Arduino_JLF_BB7200_SE**
- La BB6700 de Yohan, répertoire : **PDC_Arduino_JLF_BB67000_Y**

Pour adapter ce programme à d'autres locomotives ou à d'autres simulateurs, il faut modifier les programmes "**Data_Recues_du_PC_TSCRJI_xx.ino**" et "**Data_Envoyees_vers_PC_TSCRJI_xx.ino**" qui font l'interface avec l'ordinateur.

Le programme principal "**PDC_Arduino_JLF_xx.ino**" assure la couche matérielle.

Il met à jour les tableaux "i2c_inX_val[]" qui reflètent l'état des entrées.

Et quand l'on renseigne les tableaux "i2c_outX_val[]", il active les sorties en conséquence.

Il n'a pas besoin d'être modifié, et il est commun pour toutes les locomotives.

Dans le répertoire "...\\PDC_Arduino_JLF_xx", on doit avoir les fichiers :

- PDC_Arduino_JLF_xx.ino
- Data_Recues_du_PC_TSCRJI_xx.ino
- Data_Recues_du_PC_TSCRJI_xx.ino
- LiquidCrystal_I2C_W1.cpp
- LiquidCrystal_I2C_W1.h

Pour lancer l'environnement IDE Arduino, cliquer sur "**PDC_Arduino_JLF_xx.ino**".

Ajouter les librairies suivantes de Rob Tillaart et pour le joystick, par le gestionnaire de librairies :

- URL: https://github.com/RobTillaart/MCP23017_RT (V0.8.1)
- URL : https://github.com/RobTillaart/PCA9685_RT (V0.7.1)
- URL : https://github.com/RobTillaart/I2C_EEPROM (V1.9.2)
- URL : <https://github.com/MHeironimus/ArduinoJoystickLibrary> (V2.1.1)

ou depuis les *.zip fournis. Menu : Croquis > Inclure une bibliothèque > Ajouter la bibliothèque .ZIP... >

- MCP23017_RT-master.zip
- PCA9685_RT-master.zip
- I2C_EEPROM-master.zip
- ArduinoJoystickLibrary-master.zip

Programmation de l'Arduino et dépannage

Avant la mise sous tension, vérifier au minimum, pas de court-circuit entre :

- Le + 5 Volts, le + 5 Volts de puissance et le +3,3 Volts.
- Le + 5 Volts et le + 5 Volts de puissance.
- Le + 5 Volts et le + 5 Volts de puissance et la masse.

Alimenter la plaque en 12 Volts. On doit avoir une consommation de moins de 100 mA.

Programmer l'Arduino depuis l'interface IDE.

Il ne faut pas brancher maintenant le port usb Natif qui émule un clavier, à l'ordinateur. Si on le branche, la programmation de la carte n'est pas possible.

Si il y a un problème de compilation ou erreur 'keyboard.h', installer la bibliothèque : Arduino SAM boards (32-bit arm cortex-m0). En branchant cette carte, on doit vous proposer de la télécharger. Mettre à jour les librairies déjà installées.

Une fois alimentée en 12 Volts, si l'écran LCD n'affiche pas le menu d'accueil :

- Appuyer sur le bouton Reset.

- Régler le potentiomètre de réglage de contraste sous l'écran LCD. En position usine, rien ne s'affichera.

- Vérifier que l'alimentation 12 Volts est bien branchée. Il faut que le voltmètre de la carte indique 12 Volts.

- Vérifier les connecteurs à barrettes, et mettre un coup d'aérosol à contacts. Parfois les tiges des barrettes mâles glissent et ne font plus contact.

- Vérifier si il n'y pas eut une inversion des signaux SDA/SCL.

- Mettre le 12 Volts, avant la tension sur les prises usb.

- Vérifier le câblage et l'insertion des prises.

Si les boutons du boîtier de commande ne réagissent pas essayer le menu : (4) Information Système > Touche du LCD. J'ai déjà eu plusieurs cartes Arduino DUE avec les entrées analogiques en panne.

Poser vos questions sur le forum RMF : <https://www.rmfmagazine.com/phpBB/viewforum.php?f=21&sid=2b6958827b62a3a43f687cc722793b9b>

L'installation des programmes sur l'ordinateur Windows.

Une fois la carte Arduino programmée, on branchera les deux ports de l'Arduino DUE à l'ordinateur.

1 / Le port "Programming Atmega" assure la liaison série avec de l'ordinateur.

- Quand on branche la carte Arduino, sous Windows, on la retrouvera sous le nom COMxx (Ex : COM41).

- L'ordinateur enverra des données par ce port COMxx à l'Arduino DUE.

- La vitesse de cette liaison série avec l'Arduino est de 38400 bps.

2 / Le port "Native usb Sam3x" assure l'émulation d'un clavier supplémentaire, pour envoyer des codes de touche au simulateur, comme le fait le clavier principal.

Le logiciel "Train Simulator Classic RailWorks de DTG" est installé sur l'ordinateur.

On doit configurer ce programme avec les commandes en mode "**Expert**", depuis le menu "Paramètres.

On installera la locomotive BB67000 de Yoahn disponible sur [RailSim](#).

On peut installer une autre locomotive, mais il faudra reprendre des codes de touche, dans le programme de l'Arduino DUE. Par exemple, la touche du clavier commandant de sablage peut être différente.

On installe le programme "TSClassic Raildriver and Joystick Interface V3.3.0.7". Ce programme va chercher les informations dans le programme "Train Simulator Classic", pour les exporter :

<https://forums.dovetailgames.com/threads/ts-classic-raildriver-and-joystick-interface.72488/>

Faire une sauvegarde des fichiers originaux, puis copier mes fichiers en les renommant :

- **AssignedPorts** - Fichier de référence pour **BB67000_Yohann.txt** sous le nom **AssignedPorts.txt** dans le répertoire : TSClassic Raildriver and Joystick Interface/KeysMaps/Ports/
- **ControlNames** - Fichier de référence pour **BB67000_Yohann.txt** sous le nom **ControlNames.txt** dans le répertoire : TSClassic Raildriver and Joystick Interface/Settings/

Le fichier **ControlNames.txt** contient toutes les variables que le programme d'interface va chercher dans "Train Simulator Classic RailWorks de DTG", pour l'envoyer à l'Arduino DUE.

- Le premier champ est un nom libre, pour classer les informations.
- Le deuxième champ est un nom libre pour classer les informations, en plus court. Ne pas dépasser pas 3 caractères, car ce champ est envoyé à l'Arduino, et cela prend du temps sur la liaison série à 38400 Bps.
- Le troisième champ est le plus important. C'est lui qui est la référence entre l'ordinateur et l'Arduino. Ce champ n'est pas libre. Il est imposé.

On retrouve le libellé de ce champ dans le fichier "BB 67471.txt" fourni avec la locomotive, ou quand le simulateur et l'interface ont fonctionnés, dans le fichier : TSClassic Raildriver and Joystick Interface/debug.txt.

1 / Extrait du fichier **ControlNames.txt** :

```
LEVIER=LEV=Chauffage_Train
VOYANTS=VOY=VY_Batt
MANO=MNO=RE_Anim
PWM=PWM=AMP_Moteur
```

2 / Extrait du fichier **AssignedPorts.txt** avec les mêmes variables, avec l'ajout du port série utilisé :

```
BaudRate=38400
LEVIER=LEV=Chauffage_Train=COM41
VOYANTS=VOY=VY_Batt=COM41
MANO=MNO=RE_Anim=COM41
PWM=PWM=AMP_Moteur=COM41
```

Dans le fichier **AssignedPorts.txt**, il faudra modifier le port **COM41** avec le n° de port attribué sur votre ordinateur.

Lancer le programme : TSClassic Raildriver and Joystick Interface

Dans le menu : Settings > Mode > Cocher la case ☒ **Advanced**.

Faire une extraction des données, menu : Railworks Data Extractor > Extract All Files (*C'est long*).

Dans le menu : Output Data > Update Serial Data, on retrouve les données du fichier "**AssignedPorts.txt**".

Vérifier la vitesse de transmission = 38400.

Dans le menu : Settings Edit Control Names, on retrouve le contenu du fichier "**ControlNames.txt**".

Ne pas faire d'erreur en les modifiant manuellement ces fichiers .txt. En cas d'erreur, le programme d'interface supprime des lignes dans ces fichiers. En tout cas, garder une copie.

On peut aussi remplir ces fichiers avec le programme d'interface, mais c'est plus long.

Si l'on ajoute une locomotive, faire une extraction des données, menu : Railworks Data Extractor > Extract Single Folder.

Une fois installé et configuré, l'ordinateur envoie ce genre de trames à l'Arduino :

```
<RPM : VirtualRPM : .670><Speed : SpeedometerKPH : .010> <Acceleration : Acceleration : .352> <RPM : VirtualRPM : .700> <Speed : SpeedometerKPH : .011>
<Acceleration : Acceleration : -.034> <RPM : VirtualRPM : .705> <Speed : SpeedometerKPH : .012> <Acceleration : Acceleration : .165> <RPM : VirtualRPM :
.701> <Speed : SpeedometerKPH : .011> <Acceleration : Acceleration : .046> <RPM : VirtualRPM : .698> <Speed : SpeedometerKPH : .011> <Acceleration :
Acceleration : -.172> <RPM : VirtualRPM : .690> <Speed : SpeedometerKPH : .011> <Acceleration : Acceleration : .002> <RPM : VirtualRPM : .701> <Speed :
SpeedometerKPH : .011> <Acceleration : Acceleration : -.390> <RPM : VirtualRPM : .707> <Speed : SpeedometerKPH : .011> <Acceleration : Acceleration :
.000> <RPM : VirtualRPM : .707> <Speed : SpeedometerKPH : .012> <Acceleration : Acceleration : .000> <RPM : VirtualRPM : .706> <Speed : SpeedometerKPH
: .012>
```


Le programme 'Data_Recues_du_PC_TSCRJI_Y.ino' traite les sorties suivantes :

Les sorties marquées X ne sont pas utilisées, et restent disponibles.

Les sorties OUT2 n° 8 à 15, animent une maquette de locomotive, posée sur le pupitre.

ARDUINO - SORTIES - BB67471 Yohan

Le 05/02/2025

Variables de retour d'info

Manomètre RE / Gros mano à gauche / Aiguille blanche	0	SERVO	RE_Anim
Manomètre CP / Gros mano à gauche / Aiguille rouge	1		CP
Manomètre CG / Mano au Centre	2		CGanim
Manomètre CF1 / Mano à Droite / 1 trait	3		CFanim
Manomètre CF2 / Mano à Droite / 2 traits	4		CF2anim
X	5		X
X	6		X
X	7		X
Ampèremètre Moteur de Traction 1	8		AMP_Moteur
Ampèremètre Moteur de Traction 2	9		AMP_Moteur
Voltmètre Ligne Train A / TensionLDT	10		Tension_LDT
Compte-tours / RPM	11		RPM
IV / Tachro / Compteur de vitesse	12	PWM	SpeedometerKPH
X	13		X
X	14		X
X	15		X

Voyant / LS QT1 / VY_QT1	0		VY_QT1
Voyant / LS QT2 / VY_QT2	1		VY_QT2
Voyant / LS_L1 / Autorisation Lancement MD 1	2		VY_MD1
Voyant / LS_L2 / Autorisation Lancement MD 2	3		VY_MD2
Voyant / LC_IC / Défaut charge Batterie	4	OUT 1	VY_Batt
Voyant / LS PAT / VY_PAT / Patinage des roues	5		VY_PAT
X	6		X
X	7		X
Voyant / ZP SURC / Surcharge	8		Vy_Surcharge
Voyant / Témoin alerte SAL	9		Vy_SAL
Alarme sonore / Vacma_CA	10		Vacma_CA
Alarme sonore / Vacma_RA	11		Vacma_RA
Voyant / HandBrake	12	OUT 1	HandBrake
X	13		X
X	14		X
X	15		X

Voyant / Eclairage instruments du pupitre	0		Eclairage_Pupitre
Voyant / Eclairage cabine	1		Eclairage_Cabine
X	2		X
X	3		X
X	4	OUT 2	X
X	5		X
X	6		X
X	7		X
Pour maquette	8		Fanal_Gauche
Feu blanc gauche	9		Fanal_Droit
Feu blanc droit	10		Projecteurs_1
Feu blanc projecteur	11		Feu_RougeG
Feu rouge gauche	12	OUT 2	Feu_RougeD
Feu rouge droit	13		Eclairage_Cabine
Eclairage cabine	14		Feu_RougeG2
Feu rouge gauche Cab 2	15		Feu_RougeD2
Feu rouge droit Cab 2			

Sorties PCA9685, courant maxi :
- 25 mA entre une sortie et le + 5 Volts
- 10 mA entre une sortie et la masse.

Servomoteur - Impulsion de 700 µs à 2300 µs
PWM - Fréquence = 62 Hz

Les fonctionnalités et la configuration de l'interface

Mon montage reprend les commandes disponibles depuis un clavier.

Pour la locomotive BB67000 de Yohan, voici la liste des codes de touches à envoyer au simulateur.

Fonctions	Nom en Cabine	Touche Clavier Azerty	Sur Arduino
Batterie	Sectionneur Batterie	Ctrl+N	Boitier de Cmd
Boite à leviers Poste 1	Boite à Leviers Poste 1	W	Pupitre
Commutateur Éclairage	Commutateur Éclairage	Shift+E	Boitier de Cmd
BP(L)D1	Lancement Moteur 1	P	Pupitre
BP(L)D2	Lancement Moteur 2	Shift+P	Pupitre
BP Start Essai	Lancement Essai	Ctrl+P	
BP(A)D1	Arrêt Moteur 1	Y	Pupitre
BP(A)D1	Arrêt Moteur 2	Shift+Y	Pupitre
BP Stop Essai	Stop Essai	Ctrl+Y	
Éclairage Cabine 1	Éclairage Cabine 1	J	Pupitre
Éclairage Pupitre	Éclairage Pupitre	Ctrl+J	Pupitre
Z-IS-VA	Isolement VACMA	Ctrl+K	
Essais VACMA 1	Essais VA	K	Pupitre
Essais VACMA 2	Essais VA	Shift+K	
Essuie-glaces	Essuie-glaces	V	Pupitre
Chauffage Train poste 1	Chauffage Train	N	Pupitre
Chauffage Train poste 2	Chauffage Train	Shift+N	
Accélération Diesel poste 1	Accélération Diesel	I	Pupitre
Accélération Diesel poste 2	Accélération Diesel	Shift+I	
Sablage	Sablage	X	Pupitre
Projecteurs Poste 1	Projecteurs 1	H	Pupitre
Projecteurs Poste 2	Projecteurs 2	Shift+H	
Réarmement QT	QT	O	Pupitre
Inverseur de traction	MPJ	Z-S	Pupitre
Cerclo Poste 1	Traction	Q-D	Pupitre
Cerclo Poste 2	Traction 2	Shift+Q/Shift+D	
Cerclo Veille	Veille active	Espace	Pupitre
Feux Rouges Gauche P2	Feu Rouge Gauche	F	
Feux Rouges Droit P2	Feu Rouge Droit	Ctrl+F	
Z-IS-RS	Isolement RS	+(Pavé_Num)	
Z-IS-KVB	Isolement KVB	F11	
BP(AC)SF	Acquittement Signal Fermé	Entrée(Pavé_Num)	Pupitre
Frein de Train	MPF	M/ù	Pupitre
Frein Direct	FD	^-_\$	Pupitre
Frein à Vis	Frein à main	Ctrl+B	Boitier de Cmd
Compresseur Poste 1	Compresseur Auto	C	Pupitre
BP(URG)	Urgence	Retour	Pupitre
Desserrage Grand Débit	Grand Débit	L	Pupitre
Sifflet	Sifflet	Page up /down	Pupitre
Z-HM	Isolement moteur de Traction		
Fermeture Porte CPT		U	
Lanterne de Bord	--	Ctrl+L	
Message Tension BA	--	-(Pavé Num)	
Message Longueur/Masse Train		F10	
Fanal Blanc Gauche	Fanal Gauche	R	Pupitre
Fanal Blanc Droit	Fanal Droit	Ctrl+R	Pupitre

Si une commande du pupitre n'a pas de raccourci clavier, elle n'est pas utilisée.

La vitesse d'échange avec la carte Arduino DUE est de 38400 bps.

Le programme 'Data_Envoyée_vers_le_PC_TSCRJI_xx.ino'

Quand une des entrées sur les cartes d'entrée 1, 2 ou 3 change d'état, le programme la traite.

Il envoie alors un code de touche au simulateur sur l'ordinateur. Par exemple les lettres 'Q' et 'D' pour le Cerclo.

Il faut convertir le code des touches d'un clavier azerty vers un clavier qwerty.

En branchant le port "Native usb Sam3x" de l'Arduino sur l'ordinateur, il émule automatiquement un clavier. Il n'y a rien d'autre à faire.

Attention, quand on manipule le pupitre et que "Train Simulator Classic" n'est pas lancé, des codes de touches vont être envoyés à l'ordinateur, comme si on les tapait sur le clavier.

Quand une entrée a bougé, on récupère la valeur de l'entrée $X = \text{bitRead}(\text{i2c_in1_val}, X) = 0$ ou 1 . ($1 = \text{ON}$).

On envoie un code de touche au simulateur.

La routine **tch_env**(Normal, Majuscule ou Control, Code de la touche, Durée d'appui en msec) assure l'envoi en tâche de fond des codes des touches.

Dans ce programme, on traite les 16 entrées de la carte in1, de la carte in2 puis de la carte in3.

On traite les boutons poussoir, en envoyant un code de touche "PRESS" à la fermeture du contact, puis un code de touche "RELEASE" à l'ouverture du contact.

Dans ce cas, on utilise les codes "PRESS" et "RELEASE" à la place d'un délai en msec.

```
if (x == 1) { if(bitRead(i2c_in1_val, x) == 1) tch_env(NORMAL, 'x', PRESS); else tch_env(NORMAL, 'x', RELEASE); } //  
x      Bouton poussoir Sable.  
  
else if (x == 2) { if(bitRead(i2c_in1_val, x) == 1) tch_env(NORMAL, '.', PRESS); else tch_env(NORMAL, '.', RELEASE); } //  
. pour / Desserrage Grand Débit.  
  
else if (x == 6) { if(bitRead(i2c_in1_val, x) == 1) tch_env(NORMAL, 'c', PRESS); else tch_env(NORMAL, 'c', RELEASE); } //  
c      Petit levier Frein du train en avant.
```

On traite les interrupteurs en envoyant un code de touche, à chaque changement d'état de l'entrée.

Dans ce cas, on garde en mémoire l'état de l'interrupteur dans le champ : **i2c_in1_val_ref**.

Le programme de réception des données, va mettre à jour cet état depuis le simulateur.

S'il y a un écart, on enverra de nouveau un code de touche.

Comme cela, l'état du simulateur est toujours conforme à celui du pupitre.

```
if (x == 3) { tch_env(NORMAL, KEY_BACKSPACE, duree_touche_100);  
bitWrite(i2c_in1_val_ref, x, bitRead(i2c_in1_val, x)); } // 'KEY_BACKSPACE' Frein d'urgence.  
  
else if ((x == 4) && (bitRead(i2c_in1_val, x) == 1)) { tch_env(NORMAL, ',', duree_touche_200); bitWrite(i2c_in1_val_ref, x,  
bitRead(i2c_in1_val, x)); } // , pour . Grand levier Frein direct en avant. 200 msec.  
  
else if ((x == 5) && (bitRead(i2c_in1_val, x) == 1)) { tch_env(NORMAL, 'm', duree_touche_100); bitWrite(i2c_in1_val_ref, x,  
bitRead(i2c_in1_val, x)); } // m pour ? Grand levier Frein direct en arrière.  
  
else if ((x == 12) && (bitRead(i2c_in1_val, x) == 1)) { tch_env(NORMAL, 'w', duree_touche_300); bitWrite(i2c_in1_val_ref, x,  
bitRead(i2c_in1_val, x)); } // w pour z Inverseur de traction, en avant. 300 msec.
```


Pour les interrupteurs à plusieurs positions, il manque un contact pour savoir si on est en position centrale.

Dans ce cas, si les interrupteurs sont tous ouverts, on remet automatiquement en place le levier.

```
if ((x == 4) || (x == 5)) && ((bitRead(i2c_in1_val, 4) == 0)) && (bitRead(i2c_in1_val, 5) == 0)) {

    if (bitRead(i2c_in1_val_ref, 4) == 1) { tch_env(NORMAL, 'm', duree_touche_100); } // m pour ?      Grand levier Frein
    direct en arrière. 100 msec.

    else if (bitRead(i2c_in1_val_ref, 5) == 1) { tch_env(NORMAL, ',', duree_touche_200); } // , pour .      Grand levier Frein
    direct en avant. 200 msec.

    bitWrite(i2c_in1_val_ref, 4, 0); bitWrite(i2c_in1_val_ref, 5, 0);
}
```

Si il y a des potentiomètres, comme pour la lecture du Cerclo, on les traite en envoyant des séquences de durée calibrées de touches.

C'est subtil à programmer, car il faut définir par des essais successifs, la durée à attribuer aux touches pour obtenir un mouvement de 0 à 100%.

Suivant les appareils, on peut choisir d'utiliser 8, 16 ou 32 pas.

On envoie une touche pendant 25 msec, pour un faible déplacement, à 200 pour un grand déplacement.

```
if (millis() > lect_can_milli) {                                // Pour ne pas passer 25000 fois par seconde dans ce code. On économise du
cpu par boucle.

    lect_can_milli = millis() + 100;                            // On passe ici toutes les 100 msec.

    regul_val = analogRead(pin_regul_broche);                  // 0 à 1023.

    if ((regul_val - regul_val_1023 > 10) || (regul_val - regul_val_1023 < -10)) { // Un peu d'hystérésis, pour que la valeur
    ne bouge pas sans arrêt.

        regul_val_1023 = regul_val;
    }

    regul_val = regul_val_1023 >> 5;                            // >>5 : 0 à 31 = 32 Pas de régulation.

    if (regul_val != regul_val_ref) {

        if (regul_val == 31) { tch_env(NORMAL, 'a', 200); regul_val_ref = regul_val_ref + 7; } // a pour q
        Traction : (+) <-- On arrive en butée.

        else if (regul_val == 0) { tch_env(NORMAL, 'd', 200); regul_val_ref = regul_val_ref - 7; } // d
        Traction : --> (-) On arrive en butée.

        else if (regul_val >= regul_val_ref + 4) { tch_env(NORMAL, 'a', 100); regul_val_ref = regul_val_ref + 4; } // a pour q
        Traction : (+) <--

        else if (regul_val <= regul_val_ref - 4) { tch_env(NORMAL, 'd', 100); regul_val_ref = regul_val_ref - 4; } // d
        Traction : --> (-)

        else if (regul_val > regul_val_ref) { tch_env(NORMAL, 'a', 25); regul_val_ref++; } // a pour q
        Traction : (+) <--

        else if (regul_val < regul_val_ref) { tch_env(NORMAL, 'd', 25); regul_val_ref--; } // d
        Traction : --> (-)

        if (regul_val_ref > 31) regul_val_ref = 31;

        else if (regul_val_ref < 0) regul_val_ref = 0;

        ic2_in_comp_ref_milli = ic2_in_comp_ref_milli + duree_touche_200; // Si le cerclo a bougé sur le pupitre, on attendra un
        peu plus avant de vérifier les valeurs du simu.
    }
}
```

Si l'on a branché un boîtier KVB, on récupère l'information d'un bouton appuyé sur le KVB, pour l'envoyer au simulateur.

```
if (Serial3.available() > 0) {

    char car_lu = Serial3.read();
}
```

```

if (car_lu == 'A') { tch_env(KEY_LEFT_SHIFT+KEY_LEFT_CTRL, 'v', duree_touche_200); } // Appui sur le bouton [VAL] du
KVB. 200 msec obligatoire

if (car_lu == 'E') { tch_env(NORMAL, KEY_KP_ENTER, duree_touche_200); } // Appui sur le bouton [SF] du
KVB. 200 msec obligatoire

```

On choisit les touches à envoyer lorsque l'on appuie sur les boutons du boîtier de commande.

```

if (int_0_val == 1) { // Si l'interrupteur "ENVOI" est sur 'ON' = 0, on inhibe l'envoi des touches au PC.

    if ((bp_1_cpt == 0xFFFF) && (bp_1_val == 0)) { tch_env(KEY_LEFT_CTRL, 'n', duree_touche_100); bp_1_val = 1; } // ^n :
    Sectionneur batterie.

    if ((bp_2_cpt == 0xFFFF) && (bp_2_val == 0)) { tch_env(KEY_LEFT_SHIFT, 'e', duree_touche_100); bp_2_val = 1; } // MajE :
    Commutateur éclairage.

    if ((bp_3_cpt == 0xFFFF) && (bp_3_val == 0)) { tch_env(KEY_LEFT_CTRL, 'n', duree_touche_100); bp_3_val = 1; } // n :
    Frein à main.

```

Le programme 'Data_Recues_du_PC_TSCRJI_xx.ino'

Il reçoit les trames de l'ordinateur sous la forme : ><Speed : SpeedometerKPH : 150.0>

Une fois la trame complète reçue, il renseigne les trois chaînes de caractères suivantes :

- buffer_rx[0] = Nom du groupe de données (Texte).
- buffer_rx[1] = Nom de la donnée (Texte).
- buffer_rx[2] = Valeur au format texte (Nombre entier ou avec virgule).

Le champ buffer_rx[0] ne me sert pas, c'est pour cela que je mets que 3 caractères, pour réduire le temps de transmission de la trame.

Le champ buffer_rx[1] est le 3ème champ du fichier "AssignedPorts.txt".

Exemple : Chauffage_Train, VY_Batt, RE_Anim, AMP_Moteur.

Le champ buffer_rx[2] est la valeur de la variable. C'est du texte ou des nombres transmis au format texte. Ça peut être des entiers ou des nombres flottants.

Pour les sorties de type SRV, on renseigne un tableau "i2c_srv_val", avec obligatoirement une valeur comprise entre **0 et 255** !

Pour les sorties de type PWM, on renseigne un tableau "i2c_pwm_val", avec obligatoirement une valeur comprise entre **0 et 255** !

Exemple pour le servomoteur d'un manomètre.

Si le manomètre a un cadran de 0 à 9 Bars, mais que la valeur maximum est de 5,5 Bars, il faut utiliser ces 5,5 bars pour faire le calcul d'échelle. C'est la bonne manière pour obtenir la meilleure résolution possible.

On indique dans **i2c_srv_val[x]** et **bitWrite(i2c_srv_maj, x, 1)**, le numéro de sortie **x** du servomoteur.

```

if (!strcmp(buffer_rx[1], "RE_control")) {

    buffer_rx_fvaleur = atof(buffer_rx[2]); // Conversion du dernier champ : Ascii-> Flottant. Pas de contrôle ! Si
    Nok valeur = 0 !

    buffer_rx_fvaleur = 46.54*buffer_rx_fvaleur; // 5.5 Bars => buffer_rx_fvaleur = 256 au maximum (Coéf = 256/5.5). Une
    seule multiplication en flottant, pour économiser le cpu.

    buffer_rx_valeur = int(buffer_rx_fvaleur); //

    buffer_rx_valeur = min(buffer_rx_valeur, 255); // Valeur <= 255.

    i2c_srv_val[0] = max(buffer_rx_valeur, 0); // Valeur positive !

    bitWrite(i2c_srv_maj, 0, 1);

```

```
}
```

Exemple pour une sortie.

Attention pour des interrupteurs, souvent la valeur reçue passe par plusieurs valeurs intermédiaires. 0, 0.12, 0.35, 0.70, 1.00. C'est pour cela que je teste les valeurs avec la routine : **buffer2_compare_float_maj_out**(out1, Numéro de la sortie).

Cette routine compare la valeur reçue à 0,02 et 0,98, pour savoir si c'est un 0 ou un 1.

Si l'on est sûr que la valeur envoyée est 0 ou 1, on utilisera la routine **buffer2_compare_int_maj_out**().

Sinon, on utilisera la routine : **buffer2_compare_float_maj_out**().

```
else if (!strcmp(buffer_rx[1], "LS_DJ_control")) buffer2_compare_int_maj_out(out1, 0); // < :LS_DJ_control:1>
Disjoncteur
```

Si l'on veut recevoir l'état d'un interrupteur, on utilise la routine **buffer2_compare_float_maj_sim**().

Cette routine met à jour la valeur de référence de l'entrée x, et déclenche une comparaison avec l'entrée physique.

```
else if (!strcmp(buffer_rx[1], "inter_DJ_control")) buffer2_compare_float_maj_sim(in2, 1); // < :inter_DJ_control:1>
Info en retour sur le Disjoncteur. La valeur passe progressivement de 0.0 à 1.00.
```

Pour les interrupteurs à plusieurs positions, il faut traiter la valeur reçue pour vérifier la bonne position des interrupteurs.

```
else if (!strcmp(buffer_rx[1], "FD_control")) { // Grand levier de frein direct à trois positions.

    buffer_rx_fvaleur = atof(buffer_rx[2]); // Obligé de passer par un nombre flottant.

    if (buffer_rx_fvaleur > 0.38) { bitWrite(i2c_in1_val_sim, 4, 0); bitWrite(i2c_in1_val_sim, 5, 1);

        bitWrite(i2c_in1_maj_sim, 4, 1); bitWrite(i2c_in1_maj_sim, 5, 1); // La comparaison et demande de mise à jour, se fera
        automatiquement toutes les xxx msec.
    }

    else if (buffer_rx_fvaleur < -0.23) { bitWrite(i2c_in1_val_sim, 4, 1); bitWrite(i2c_in1_val_sim, 5, 0);
        bitWrite(i2c_in1_maj_sim, 4, 1); bitWrite(i2c_in1_maj_sim, 5, 1); // La comparaison et demande de mise à jour, se fera
        automatiquement toutes les xxx msec.
    }
    else if ((buffer_rx_fvaleur > 0.08) && (buffer_rx_fvaleur < 0.12)) { bitWrite(i2c_in1_val_sim, 4, 0);
        bitWrite(i2c_in1_val_sim, 5, 0);

        bitWrite(i2c_in1_maj_sim, 4, 1); bitWrite(i2c_in1_maj_sim, 5, 1); // La comparaison et demande de mise à jour, se fera
        automatiquement toutes les xxx msec.
    }
}
```

Pour le KVB, une fois reçu une commande du simulateur, on la renvoie au KVB.

Le caractère placé dans **KVB_car_vers_KVB** sera automatiquement envoyé au KVB.

```
if(!strcmp(buffer_rx[0], "KVB")) {
    buffer_rx_valeur = atoi(buffer_rx[2]); // Valeur = 0 ou 1.

    if (!strcmp(buffer_rx[1], "KVB_BP_VAL_lumiere_control")) {if(buffer_rx_valeur == 1) KVB_car_vers_KVB = 'G'; else
    KVB_car_vers_KVB = 'g'; }

    else if(!strcmp(buffer_rx[1], "KVB_BP_CAR_lumiere_control")) {if(buffer_rx_valeur == 1) KVB_car_vers_KVB = 'I'; else
    KVB_car_vers_KVB = 'i'; }

    else if(!strcmp(buffer_rx[1], "KVB_BP_VIO_lumiere_control")) {if(buffer_rx_valeur == 1) KVB_car_vers_KVB = 'H'; else
    KVB_car_vers_KVB = 'h'; }
```

Test de la carte Arduino DUE

Brancher un servomoteur sur la sortie servo n° 0.

Brancher un voltmètre sur la sortie pwm n° 0 et le +5 V.

Brancher une led sur la sortie out1 n° 0 et le + 5V.

La carte Arduino DUE est déjà programmée.

1 / Mettre le montage sous tension.

Le menu d'accueil s'affiche sur l'écran LCD du boîtier de commande :

JLF - 13/02/2025

MENU: TURNER BTN

Si le menu d'accueil ne s'affiche pas correctement, appuyer sur le bouton RAZ de l'Arduino.

Régler le potentiomètre de réglage de contraste sous l'écran LCD. En position usine, rien ne s'affichera.

Il faut mettre le 12 Volts, avant la tension sur les prises usb.

Si rien ne s'affiche, appuyer sur le bouton 'Reset'.

Vérifier le câblage et l'insertion des prises.

Si les boutons du boîtier de commande ne réagissent pas essayer le menu : (4) Information Système > Touche du LCD.

J'ai déjà eu plusieurs cartes Arduino DUE avec les entrées analogiques en panne.

Brancher les deux ports de l'Arduino à l'ordinateur.

Ouvrir le bloc-notes.

Cliquer dans la fenêtre du bloc-notes.

Mettre des plots des entrées in1, in2 ou in3 à la masse.

On doit avoir des lettres qui s'affichent dans le bloc, Exemples in3_4 => 'p', in3_6 => 'y'.

Ca correspond au tableau des touches envoyées par l'Arduino.

2 / Avec le boîtier de configuration, configurer la sortie pwm n° 0 en mode inverse.

3 / Double-cliquer sur "PDC_Arduino_JLF.ino".

Dans l'interface IDE Arduino, choisir la bonne carte et le bon port série.

Lancer le moniteur série, menu : Outils > Moniteur série

Mettre la bonne vitesse = **38400 Bps**.

Sur la ligne du moniteur série, taper :

< :RE_Anim:0.00> (*valider chaque commande avec la touche [Entrée]*)

< :RE_Anim:3.00>

< :RE_Anim:6.00>

Le servomoteur doit se positionner sur trois positions différentes.

Sur la ligne du moniteur série, taper :

< :AMP_Moteur:0> (*valider chaque commande avec la touche [Entrée]*)

< :AMP_Moteur:1000>

< :AMP_Moteur:2000>

La sortie pwm doit passer de 0 à 2,5, puis 5 Volts

Sur la ligne du moniteur série, taper :

< :VY_QT1:0>

< :VY_QT1:1>

La sortie led soit s'allumer ou s'éteindre.

Attention

1 / Si l'on modifie les programmes d'entrées ou de sortie, ne pas confondre les variables :

i2c_in1_maj, i2c_in1_val, i2c_in1_val_ref, i2c_in1_val_sim, i2c_in1_maj_sim

i2c_in1_maj : Demande de traitement de l'entrée physique.

i2c_in1_val : Valeur de l'entrée physique.

i2c_in1_val_ref : Valeur mémorisée de l'entrée, une fois le code de touche envoyé au simulateur.

i2c_in1_val_sim : Valeur envoyée par le simulateur.

i2c_in1_maj_sim : Demande de vérification de l'entrée, car la valeur du simulateur vient d'être envoyée.

2 / Si l'on modifie les programmes d'entrées ou de sortie, ne pas confondre les variables :

i2c_in1_val, i2c_in2_val, i2c_in3_val

3 / Si l'on modifie les programmes d'entrées ou de sortie, ne pas confondre les variables :

i2c_in1_val, i2c_out1_val

A+