**DIFFUSION RESTREINTE**

Une image contenant Graphique, graphisme, Police, logo

Description générée automatiquement

Manuel du système d’acquisition et de contrôle du banc de l’allumeur-torche

Propulsion / Banc de test

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Document initial** | |  |  |  |
| **Version** | **A** | B | C | D |
| **DATE**  **VISAS** | 06/09/24 |  |  |  |
| **Rédacteur** | Mehdi Delouane |  |  |  |
| **Validation** |  |  |  |  |

**LISTE DE DIFFUSION**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Noms | Organisme | Nb. | Pour | | |
| Approb. | Accept. | Info. |
| Diffusion interne | | | | | |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| Diffusion externe | | | | | |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**ENREGISTREMENT DES EVOLUTIONS DU DOCUMENT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Edition** | **Date** | **Pages** | **Référence et résumé des modifications** |
| A | 06/09/24 | 21 | Document initial |

**TABLE DES MATIERES**

[I. Terminologie 5](#_Toc164449343)

[Références 6](#_Toc164449344)

[II. Guide utilisateur 7](#_Toc164449345)

[1. Introduction 7](#_Toc164449346)

[2. Description de l’électronique du banc 9](#_Toc164449347)

[3. Prise en main 11](#_Toc164449348)

[3.a. Lancement du logiciel 11](#_Toc164449349)

[3.b. Interface utilisateur 11](#_Toc164449350)

[4. Exemple d’utilisation 15](#_Toc164449351)

[Documents annexes 21](#_Toc164449352)

[Annexe A : Synoptique fluide du banc sans débitmètres (1er) et avec débitmètres (2e) 21](#_Toc164449353)

[Annexe B : Schéma de câblage 23](#_Toc164449354)

**Liste des figures**

[Figure II‑I : Interface utilisateur du logiciel de pilotage du banc d'essai 9](#_Toc164428438)

[Figure II‑II : Exemple de fichier de séquence 10](#_Toc164428439)

[Figure II‑III : Affichage du choix et du lancement d’un essai 14](#_Toc164428440)

# Terminologie

Les sigles, abréviations et acronymes utilisés dans le présent document sont listés ci-dessous.

|  |  |
| --- | --- |
| **Acronyme** | **Signification** |
| IHM | Interface homme-machine |
| IDE | Integrated Development Environment |
| ADC | Analog-digital converter |
| I2C | Inter-Integrated Circuit |
| GPIO | General-Purpose Input/Output |
| NC | Normally close |
| NO | Normally open |
| DMA | Direct Memory Access |
| UDP | User Datagram Protocol |

# Références

|  |  |  |
| --- | --- | --- |
| **ID** | **Titre du document** | **Référence** |
| REF 1 |  |  |
| REF 2 |  |  |

# Manuel d’utilisation

## Introduction

Ce manuel d’utilisation visa à expliquer le travail réalisé sur la carte microcontrôleur, la carte relai de communication BeagleBone Black et l’interface homme-machine dans le cadre du projet allumeur.

## STM32 Nucléo-F207ZG

**Objectif :**

La carte Nucléo de chez STMicroelectronics est l’élément principal du système d’acquisition et de contrôle. Elle doit principalement assurer la récupération et l’envoi de toutes les mesures des capteurs (pression, température, débit) ainsi que le contrôle de toutes les électrovannes du banc d’essai. Cette carte se code en C sur l’IDE STM32CubeIDE et son extension STM32CubeMX.

**Cahier des charges :**

* Acquisition en temps réel des différentes grandeurs physiques (temps, pression, température, débit)
* Transmission des grandeurs physiques et de l’état des vannes
  + Temps : 1 valeur à partir de l’horloge interne du microcontrôleur
  + Pression : 3 valeurs à partir d’un ADC
  + Température : 2 valeurs à obtenir via un protocole I2C
  + Débit : 2 valeurs à partir de débitmètre à impulsion
  + Etat des vannes : 7 valeurs booléennes
* Contrôle des 7 électrovannes et de la bougie d’allumage
* Acquisition et exécution automatique des séquences d’allumage
* Arrêt d’urgence

**Acquisition des grandeurs physiques et des états des vannes :**

1. Temps

L’acquisition du temps se fait très facilement grâce à la fonction HAL\_GetTick() qui renvoie le temps (en valeur entière) en milliseconde depuis l’allumage de la carte.

1. Pression

Les 3 capteurs de pression PS1, PS2 et PS3 pour les lignes C3H8, GOX et N2 respectivement renvoient une tension en 0.5V et 4.5V, cependant, les pins du microcontrôleur ne peuvent pas recevoir plus de 3.3V. L’utilisation de ponts diviseurs de tension est nécessaire (voir Figure …). Ces tensions sont alors converties grâce à un ADC. Finalement, un étalonnage a été effectué sur chaque capteur pour en déduire une fonction affine qui convertie chaque valeur ressortie par l’ADC en bar.

Etalonnage :

L’étalonnage est assez simple à réaliser. Il suffira d’une bouteille d’air suffisamment pressurisée, d’un détendeur avec une plage de pression au minimum identique avec la pression max du capteur, un raccord cruciforme pour y fixer une vanne manuelle, un manomètre et le capteur. Après avoir fixé le détendeur à la bouteille et le raccord à ce dernier, il est possible de récupérer les valeurs sur le microcontrôleur, de les comparées avec le manomètre. Il est recommandé d’effectuer des mesures à 1 ou 0.5 bar d’intervalle de zéro jusqu’à la valeur théorique que le capteur devra lire.

1. Température

Les capteurs de température fonctionnent à partir de l’effet Seebeck. Une micro-tension est générée en fonction de la température. Cette dernière est amplifiée puis récupéré grâce à un protocole I2C.

Etalonnage :

Aucun étalonnage n’est en théorie nécessaire puisque les thermocouples sont fournis déjà étalonner cependant une petite conversion reste nécessaire afin d’obtenir une valeur en degré Celsius et/ou en Kelvin.

1. Débit

Les débitmètres utilisés sont des débitmètres à impulsion. A chaque rotation de l’hélice, une impulsion est envoyée sur le fil signal. Le débit est alors déterminé en comptant le nombre d’impulsion reçu pendant un temps connu. Il est nécessaire de rajouter une résistance de tirage vers le bas (pull-down résistance) afin de limiter un potentiel bruit (valeur de résistance à déterminer).

Etalonnage :

Tout comme les capteurs de pression, les débitmètres doivent être étalonnés. Il faudra alors une pompe à eau et les raccords adaptés au débitmètre. A l’aide d’un récipient, d’une balance et d’un chronomètre, déterminer le volume d’eau déplacé par la pompe. Pour plus de précision, il est recommandé d’effectuer les mesures en changeant l’intervalle de temps (de préférence l’augmenter) ou de changer le débit de la pompe si possible. Il suffira alors d’établir une relation entre le nombre d’impulsion dans un temps donné et le débit réel. Attention : le nombre d’impulsion change en fonction de l’intervalle de temps. Bien que la précision ne soit pas grandement impactée, il est primordial de considérer ce paramètre et d’adapter la formule de conversion afin qu’elle considère ce délai d’enregistrement

*Note : Il est possible de configurer la carte STM pour que le port GPIO soit défini en mode « pull-down » évitant des soudures mais après quelques tests, il semblerait que cela ne soit pas entièrement fiable*

1. Electrovannes

Le banc d’essai de l’allumeur est composé de 7 électrovannes (6 NC et 1 NO). Leur état (ouvert/fermé) est enregistré sous une forme booléenne.

*Important : peu importe le type de vanne (NO/NC), une vanne avec l’attribution 1 signifie une vanne ouverte et l’attribution 0 signifie une vanne fermée.*

**Programmation du système d’acquisition :**

Le système d’acquisition est composé de deux fonctions principales : main() et get\_values().

**Main() :**

La fonction main se charge d’en un premier temps d’initialiser l’ensemble des paramètres nécessaires. Ensuite, la fonction va boucler sans arrêt. Dans cette boucle, la fonction get\_values() est appelée.

**Get\_values() :**

Afin de simplifier l’envoi de donnée, il est préférable d’envoyer toutes les mesures en même temps plutôt que séparer. La seule grandeur dépendante du temps est le débit (puisque le temps influe sur le nombre d’impulsion). Comme expliqué plutôt, le programme va compter le nombre d’impulsions dans un temps donné, c’est ce temps qui détermine la fréquence d’envoi des valeurs de capteurs (il faut trouver un compromis entre précision et rapidité).

Lorsque le temps écoulé dépasse ce temps seuil, toute interruption est désactivée. Les débits sont calculés et les compteurs d’impulsion réinitialisés. Les interruptions sont réactivées et l’ADC récupère les valeurs de pression via le protocole DMA (le DMA est un protocole qui n’utilise pas le CPU et donc permet une rapidité d’exécution plus importante). Finalement, les protocoles I2C déterminent les températures.

**Gestion des interruptions :**

Le microcontrôleur ne peut pas gérer plusieurs tâches en même temps ainsi, il est par exemple impossible de recevoir une commande et d’envoyer des données en simultanées. Les interruptions permettent de stopper ou de prioriser un processus. Dans le cas des débitmètres, le microcontrôleur doit enregistrer chaque impulsion grâce à un compteur tout en assurant la réception de commande. Pour palier ce problème, à chaque impulsion, le programme est interrompu temporairement afin d’actualiser les compteurs. Ce rôle de gestion est assuré par les fonctions HAL\_GPIO\_EXTI\_Callback() et EXTI9\_5\_IRQHandler(). Si EXTI9\_5\_IRQHandler associe une interruption à un port, HAL\_GPIO\_EXTI\_Callback() lui s’occupe d’appliquer le programme lié à l’interruption. Les impulsions sont détectées par le biais de port GPIO en mode « rising edge », c’est-à-dire que lorsque le port GPIO perçoit un front montant (tension supérieure à 0).

*La différence entre un GPIO input et un GPIO rising edge vient de la réaction du port. Un GPIO input cherchera à quantifier la tension alors qu’un GPIO rising edge se comporte de manière booléenne (tension = 0 🡪 0, tension > 0 🡪 1).*

\_\_disable\_irq() et \_\_enable\_irq() permettent de désactiver et d’activer les interruptions ce qui implique que si les interruptions sont désactivées, la fonction HAL\_GPIO\_EXTI\_Callback() ne pourra jamais être appelée.

Il est possible de hiérarchiser les interruptions en priorisant une interruption à une autre dans le cas ou deux interruptions auraient lieu en même temps.

**Envoi des données :**

L’envoi des données est assez simple à comprendre. La fonction main() se charge d’initialiser la communication UDP. Notamment, elle assigne une adresse IP à la carte.

Lorsque la fonction get\_values() a fini l’enregistrement des données, la fonction transmit\_sensor\_values() est alors appelée. Les valeurs de pression, enregistrées dans une liste sont converties suivant les formules d’étalonnage. L’envoi de chiffre décimal est compliqué et peu efficace, par conséquent, les données ont été multipliés par des coefficients (10, 100, 1000) afin de conserver une certaine précision.

Un caractère ou « char » enregistre toutes les données à envoyer sous la forme suivante :

Temps, PS1, PS2, PS3, TS1, TS2, FS1, FS2, bit6, bit5, bit4, bit3, bit2, bit1, bit0

Avec bit l’état de la vanne associée.

Le caractère est alors enregistré dans un buffer puis envoyé à l’adresse IP souhaitée (dans notre cas, il s’agit de l’adresse IP de la BeagleBone Black).

*En langage C, un caractère est un type d’objet servant à enregistrer des données.*

**Réception des commandes**

La carte STM se doit de recevoir quatre types de commande : l’ouverture et fermeture des électrovannes, l’allumage de la bougie, la séquence d’allumage et l’arrêt d’urgence. L’ensemble de la réception est géré par la fonction udp\_receive\_callback(). Cette fonction provient de la librairie « lwip » ce qui lui donne l’avantage d’avoir une priorité sur les autres fonctions du programme. Ainsi, lorsqu’une communication UDP est perçue, notamment en détectant la présence de données dans son buffer, le programme va automatiquement prioriser la réception et le traitement de l’information reçue.

**Ouverture et fermeture des électrovannes**

Chaque électrovanne est représentée par un booléen (1 ou 0). Il est alors possible de simplifier l’envoi d’une commande d’ouverture ou de fermeture en regroupant tous les états en un seul nombre binaire. Voici l’ordre utilisé :

EV6, EV5, EV4, EV3, EV2, EV1, EV0 soit

N\_SV151, NP\_SV031, N\_SV131, N\_SV031, IO\_SV003, IP\_SV101, IP\_SV001

*Rappel : le type de vanne n’est pas important, seul son état (ouvert/fermé) est représenté par le booléen*

Dans le cas du banc, N\_SV151 (EV6) est une vanne normalement ouverte et les autres normalement fermées. Lors du lancement du programme, on peut alors lire « 1000000 ». Si par exemple, EV2 s’ouvrait, on lirait alors « 1000100 ». A chaque changement d’état d’une vanne, l’intégralité du message binaire est envoyée. Cela évite de devoir associer chaque message a une vanne (ex : EV6,1 pour commander l’ouverture de EV6). La commande dépend juste d’un nombre binaire ou chaque bit est associé à une électrovanne. Cela permet également donc de confirmer les commandes précédentes.

Imaginons un scénario ou la vanne EV3 doit être ouverte, que l’action est considérée validée mais que ce n’est pas le cas à la suite d’une erreur. Si par la suite, la vanne EV5 doit se fermer, la nouvelle commande contiendra toujours l’ouverture de EV3(bit n°4 défini à 1), ainsi, le code a une deuxième chance pour ouvrir EV3.

**Contrôle de la bougie d’allumage**

La bougie permet d’allumer le mélange propane-oxygène. Pour assurer une bonne combustion, la bougie doit alterner à haute fréquence entre une étincelle et l’absence d’un étincelle. Pour permettre cela, il suffit d’associer un port GPIO pour ouvrir et fermer un relai à la fréquence souhaitée. Le contrôle se fait alors par le biais d’un « flag », qui autorise ou non une section de code à être réalisée.

**Arrêt d’urgence**

L’arrêt d’urgence est primordial puisqu’il doit être recevable à tout moment. Lorsque l’arrêt d’urgence est déclenché, un « flag » est activé pour interrompre toute action et réalisée une séquence de sureté. Durant cette séquence, toutes les vannes sont fermées puis la vanne de purge est ouverte (10ms après) pendant 3s puis cette vanne se referme.

**Séquence d’essai**

La séquence d’essai permet de définir l’action d’une vanne automatiquement, à un temps donné. Le fichier qui la définit prend cette forme :

Nombre binaire à 7 bits

EV0, t1, t2, t3, …

EV1, t4, t5, t6, …

EV2, t7, t8, t9, …

…

La première ligne permet de définir pour les vannes une position de départ. Pour chaque vanne, une suite de chiffre permet de spécifier les temps (en ms) à partir duquel elle inversera l’état de la vanne. Considérons cette séquence pour 3 vannes

101

EV0, 1000, 2000

EV1, 0

EV2, 3000

Elle se traduit donc par l’ouverture de EV0 et EV2. 1 seconde après le début de la séquence, EV0 se ferme puis une nouvelle seconde plus tard (soit 2s depuis le début de la séquence), elle se réouvre. Pour finir, EV2 se ferme 3s après le début de la séquence.

Informations supplémentaires :

Si une ligne ne contient qu’un « 0 », la vanne en question ne changera jamais d’état par rapport à son état initial (défini par le nombre binaire). Dans le cas précédent, EV1 reste fermé durant l’entièreté de l’essai. Attention, si une ligne contient un « 0 » mais est suivi d’autres temps, alors, la vanne ne changera jamais d’état peu importe les temps indiqués après.

Les temps sur une même ligne doivent impérativement être différent et dans un ordre croissant. En revanche, plusieurs lignes peuvent avoir un temps en commun. Dans ce cas, au temps indiqué, chacune des vannes inversera son état.

Dans le cas ou une ligne de séquence se termine, la vanne ne changera jamais d’état même si la séquence n’est pas encore finie. Pensez donc à confirmer une position de fin pour chaque vanne.

**Programmation :**

Lorsqu’une commande est reçue, la fonction udp\_receive\_callback() vérifie l’existence du message puis elle vérifie son contenu. Si le message remplit l’un des critères ci-dessus, le code associé est exécuté et un message de confirmation est envoyé à l’ordinateur.

Si le message reçu est « STOP », cela signifie un arrêt d’urgence. Un flag est alors levé. La boucle de la fonction main() peut alors exécuter la fonction emergency() qui applique alors le protocole d’arrêt d’urgence (l’état des vannes est également actualisé). Une fois le protocole appliqué, le flag est baissé.

Si le message reçu est « SP », cela indique que la bougie doit fonctionner. De la même manière, un flag est levé permettant à la boucle du main() d’appeler la fonctionspark() qui a pour rôle d’allumer puis d’éteindre la bougie en continu.

Si le message reçu est « UNSP », cela indique que la bougie doit cesser de fonctionner. Le flag est alors baissé, la bougie cesser de créer des étincelles.

Si le message reçu a une longueur de 7, cela indique un changement dans la position des vannes. Ce nombre binaire peut provenir d’un contrôle direct d’une vanne ou de la séquence d’essai. Dans les deux cas, ce nombre est décomposé en bit. Chaque bit agit alors sur sa vanne respective pour l’ouvrir ou la fermer.

Enfin, si aucune des conditions ci-dessus n’est rempli et que le message a une longueur supérieure à 7, alors le message est une séquence. Le message est alors reçu sous cette forme (pour l’exemple donné plus tôt).

E, 1000, 2000, E, 0, E, 3000

Dans cette situation, les flags de la bougie et de la séquence sont activés. La fonction parseAndStore() est alors appelé. Cette dernière va décomposer le message dans plusieurs listes. Le terme E indique le début d’une nouvelle liste, les valeurs suivantes sont alors enregistrées dedans. Ainsi, le résultat du message ci-dessus se traduit par

List0=[1000,2000]

List1=[0]

List2=[3000]

Lorsque le découpage est terminé, un temps de référence « sequenceTime » est défini pour indiquer le début de l’essai. La boucle main va alors lancer les fonctions spark() et sequence(). La fonction sequence() va définir un temps « previousTick » puis en calculant le différentiel de temps entre previousTick et sequenceTime, il est possible d’obtenir le temps écoulé depuis le début de l’essai. Dans chacune des listes, si le temps écoulé est supérieur à la première valeur de la liste et que cette première valeur est différente de 0 alors la vanne change d’état. Son nouvel état est actualisé en modifiant la valeur du bit associé. Enfin, la fonction removeProcessedValue() est appelée. Cette dernière retire la première valeur de la liste. Enfin, si toutes les premières valeurs des listes sont des zéros, les flags de la séquence et de la bougie sont baissés. Enfin, si jamais au cours du processus, le flag d’arrêt d’urgence s’active, les flags de la bougie et de la séquence sont baissés et l’entièreté du processus décrit est annulé pour prioriser l’exécution du protocole d’arrêt d’urgence.

*Informations complémentaires :*

*Toutes les autres fonctions présentent dans le code et qui ne sont pas cités sont des fonctions présentes par défault dans le code. Elles proviennent du paramétrage de la carte grâce à STM32CubeMX.*

## BeagleBone :

La BeagleBone assure la connexion entre l’ordinateur et le microcontrolleur

## Interface Homme-Machine :

**Objectif :**

L’interface homme-machine doit servir de support visuel dans le but de simplifier les essais. En plus de fournir un apercu simplifié de la fluidique du banc, il doit pouvoir permettre le contrôle des vannes ainsi qu’afficher leur état actuel, il doit pouvoir lancer une séquence d’allumage, interrompre une séquence par le biais d’un bouton emergency et enfin, un affichage en temps réel des données des différents capteurs. Le code principal est rédigé en Python mais plusieurs fichiers externes (.txt, .csv, .png) sont nécessaires à son fonctionnement

La carte Nucléo de chez STMicroelectronics est l’élément principal du système d’acquisition et de contrôle. Elle doit principalement assurer la récupération et l’envoi de toutes les mesures des capteurs (pression, température, débit) ainsi que le contrôle de toutes les électrovannes du banc d’essai. Cette carte se code en C sur l’IDE STM32CubeIDE et son extension STM32CubeMX.

**Cahier des charges :**

* Affichage du synoptique
* Récupération en temps réel des différentes grandeurs physiques et paramètres (temps, pression, température, débit, états des vannes)
* Affichage graphique des grandeurs physiques
  + Pression : 1 graphique représentant les trois valeurs de pression
  + Température : 1 graphique représentant les deux valeurs de température
  + Débit : 1 graphique représentant les deux valeurs de débit
* Contrôle des 7 électrovannes et de la bougie d’allumage
* Affichage de l’état des électrovannes et de la bougie
* Sélection, affichage et envoi des séquences d’allumage
* Arrêt d’urgence
* Fermeture du programme

Le programme est décomposé en 4 classes : Worker, Ui\_MainWindow, MultiRealTimePlot et CountdownWidget.

**Paramètres :**

Plusieurs paramètres peuvent être définis pour spécifier les conditions de test.

Le premier paramètre est un booléen pour informer si un microcontrôleur est connecté.

Le deuxième spécifie le nom du fichier d’écriture. Il s’agit également d’un booléen ou « 1 » indique que le csv est daté avec l’heure et le jour de l’essai tandis que « 0 » indique l’utilisation d’un csv « poubelle » (chaque donnée de chaque essai est alors écrit dans ce même csv)

Le troisième paramètre défini la fréquence d’actualisation des graphiques

Les deux paramètres suivants définisset les adresses IP et ports de connexion

Le paramètre suivant définit l’état initial de chaque vanne au lancement de l’essai.

Ce paramètre spécifie le temps du décompte avant le lancement de la séquence.

**Worker :**

La classe Worker se charge de récupérer l’intégralité des messages envoyés par la STM Nucléo. Cette classe fonctionne indépendamment du reste du programme grâce au multithreading. Le multithreading facilite donc l’envoi et la réception simultanés d’informations. La classe est créée est lancé par Ui\_MainWindow (classe principale du programme) dès le lancement du programme.

Si aucune carte n’est connectée, le Worker créée des valeurs à partir de fonctions sinus, les envois au Ui\_MainWindow puis les écrits dans un fichier csv.

Si une carte est connectée, le Worker lit le message recu, si le message est inférieur ou égal à 5, il s’agit d’un message de confirmation. Si le message est supérieur à 5, il s’agit de données à enregistrer. Ce message est alors traité, envoyé à la classe principale et écrit dans le csv.

Dans chaque cas, le message est envoyé grâce à un PyQtSignal qui est le seul moyen de communication entre le Worker et Ui\_MainWindow.

**Ui\_MainWindow :**

Les deux premières fonctions (setupUi et retranslateUi) génèrent et placent l’intégralité des objets (fenêtre, image, label, bouton, etc). Pour plus d’informations à ce sujet, il est conseillé de lire les commentaires du code.

Avant même la création de la fenêtre, l’état des vannes (défini dans les paramètres) est envoyé à la carte. La fenêtre est ensuite créée avec le synoptique simplifié (voir Figure…). Le logo, les boutons, labels (noms des vannes, leur état), cadres et un tableau (qui récapitule l’état de chaque vanne) sont placés. Les fenêtres adjacentes comme celle des graphiques sont créées mais cachées. Chaque bouton est connecté à une fonction lors de sa création.

Open\_valve/close\_valve :

Ces deux fonctions ont essentiellement le même rôle et donc même fonctionnement. Elles permettent de contrôler une vanne en modifiant le nombre binaire à 7 bits. Ce nouveau nombre est envoyé à la STM via la fonction send\_command qui attend la réception d’une confirmation.

*La communication Worker🡪Ui\_MainWindow est possible grâce à un PyQtSignal mais dans l’autre sens, il faut passer par un Event.*

Une fois la confirmation réceptionnée, l’affichage de l’état de la vanne change en accord avec la sélection. Pour celay, la fonction update\_valve\_status est appelée. Elle reçoit l’identification de la vanne et change ses objets associés en accord avec la commande.

Send\_spark :

Cette fonction permet d’activer ou désactiver le fonctionnement de la bougie. Elle envoie simplement le message « SP » ou « UNSP » et attend une confirmation pour modifier le visuel du bouton.

Emergency :

Cette fonction permet d’enclencher l’arrêt d’urgence. Elle envoie simplement le message « STOP » et attend une confirmation. Une fois la confirmation recue, elle bloque l’utilisation de tous les boutons en attendant la fin de la séquence d’arrêt.

Update\_str :

Cette fonction reçoit le PyQtSignal du Worker est affiche les données sur le synoptique en temps réel.

Update\_combobox/view\_sequence/plot\_valve\_state :

Ces fonctions offrent le choix de la sequence de test. Il est possible de visualiser la séquence sur un chronographe.

Launch\_engine/activate/update\_timer:

Ces fonctions permettent de lancer la séquence d’essai et de suivre son déroulé grâce à un timer et une actualisation de l’état des vannes.

Launch\_plot :

Elle permet de lancer la fonction MultiRealTimePlot et donc d’afficher les graphiques.

Close\_all\_program :

Cette fonction a pour but d’éviter la fermeture accidentelle de l’IHM en recréant un bouton fermeture.

**MultiRealTimePlot :**

Cette classe gère l’affichage en temps réel des graphiques de trois grandeurs. La fréquence d’actualisation dépend du paramètre mentionné plus tôt. L’objectif de cette fenêtre est d’afficher en temps réel chaque grandeur, de pouvoir sélectionner la longueur de l’axe « temps » et de sélectionner les courbes à afficher ou cacher.

RTP1 :

Cette fonction génère les différents objets correspondant à la pression notamment les checkboxes pour chaque courbe et les boutons de changement de l’axe « temps ». RTP2 et RTP3 fonctionne de la même manière mais s’occupe de la température et du débit.

Update\_plot :

La fonction update\_plot va scanner le csv écrit par le Worker en permanence et enregistrés les données dans un array. Chaque liste présente dans l’array correspond à chaque ligne du csv. Récupérer la première valeur de chaque liste revient à enregistrer les valeurs de temps.

Chaque checkbox se voit alors attribué une courbe qui s’affiche seulement si la case est cochée.

Set\_x\_interval1 :

Cette fonction permet de changer la taille de l’axe « temps » pour n’afficher que les 10 dernièrs secondes, 30 dernières secondes, la dernière minute et toutes les valeurs. Set\_x\_interval2 et set\_x\_interval3 remplissent le même rôle pour les deux autres grandeurs.

**CountdownWidget :**

Cette classe a pour simple but de générer un affichage dit 7-segment display. Lorsque le compteur atteint 0, la classe envoie un message à Ui\_MainWindow qui envoie la séquence d’allumage à la carte pour lancer l’essa.

## Description de l’électronique du banc

L’électronique du banc s’articule autour des éléments suivants :

* 3 capteurs de pression ;
* 2 débitmètres ;
* 2 thermocouples
* 7 électrovannes (6NC et 1 NO) ;
* 1 bougie d’allumage
* 1 carte électronique STM32 F207ZG ;
* 1 carte électronique Beable Bone Black ;
* 1 poste de contrôle (ordinateur).

Les capteurs de pression permettent de connaître la pression des fluides à l’intérieur du banc. Ces mesures sont effectuées conformément au synoptique fluide (Annexe A). Les mesures des capteurs sont récupérées, adaptés (passage de 0.5-4.5V à 0-3.3V) et traitées directement par la carte électronique STM32 sur des pins analogiques de l’ADC1 (ADC1\_IN3, ADC1\_IN10 et ADC1\_IN13) pour les mesures de pression.

Les débitmètres sont basés sur un capteur à effet hall et une petite turbine. Le capteur à effet Hall permet de détecter les passages des pales de la turbines. Ce capteur émet une pulsation à chaque passage de pale. Ces pulsations sont détectées par les pins GPIO (PB8 et PB9). Suivant le nombre d’impulsion détecté dans un délai de temps connu, il est possible de déduire le débit du fluide.

Un thermocouple est un capteur de température qui fonctionne sur le principe de l'effet Seebeck. Il est composé de deux fils métalliques différents, soudés ensemble à une extrémité pour former une jonction. Lorsque cette jonction est exposée à une température, elle génère une tension électrique proportionnelle au différentiel de température entre cette jonction (jonction chaude) et une jonction de référence (jonction froide) située à l'autre extrémité des fils. Cette tension peut être mesurée et convertie en une valeur de température grâce à des tables de référence ou des appareils de mesure électroniques.

Les débitmètres et les capteurs de pression sont alimentés en parallèle depuis une pin « 5V » de la carte. Les thermocouples n’ont pas besoin d’être alimenté du fait de leurs fonctionnements.

La bougie d’allumage est branchée suivant le schéma suivant (schéma de câblage de la bougie).

Les électrovannes sont pilotées par la carte STM grâce à des relais 5V. Ces relais sont alimentés depuis le bloc d’alimentation 24V du banc. Un convertisseur de tension permet de passer à une tension de 5V. Le signal de commande est transmis depuis des pins GPIO (PF13, PE9, PE11, PF14, PE13, PF15, PG14). Les électrovannes sont raccordées en Normally Open (NO) ce qui signifie que lorsque le signal de commande est actif, le relais se ferme. A l’exception du relai connecté au pin PE13, quand le relais se ferme, la vanne s’ouvre. Le relai du pin PE13 contrôle une vanne NO, la fermeture du relai va donc fermer la vanne. Les électrovannes sont alimentées en parallèle en 24Vdc. Les connecteurs des électrovannes présentent un troisième fil n’étant pas utilisé dans notre cas. Le contrôle des vannes se fait par l’intermédiaire de nombre binaire ou chaque bit correspond à un relai. Un bit portant la valeur « 1 » va envoyer un signal au relai et donc le fermer. Inversement, une valeur « 0 » signifie l’absence de signal donc un relai ouvert. Ce nombre binaire est traité par la carte qui renvoie un message de validation « OK » pour confirmer à l’interface le changement d’état.

Il estimportant de noter que chaque pin possède un rôle particulier. En effet, bien que les capteurs soient identiques, les mesures récupérées par la carte sont traitées afin d’être ordonnées. Par exemple, si le pin du capteur PS1 est échangé avec celui du capteur PS2, les valeurs affichées sur l’IHM seront inversées. **Il est donc essentiel de se référer au schéma de câblage (voir Annexe B) et de le respecter lors du montage des capteurs**. Pour assurer un bon branchement, un étiquetage est présent sur les câbles sortant du boîtier électronique permettant d’indiquer quel capteur doit être branché sur chaque câble.

L’entièreté des échanges entre la carte STM et le PC se font via la Beagle Bone. La connexion STM-Beagle Bone est effectué via un câble Ethernet et la connexion PC-Beagle Bone se fait sans fil.

## Prise en main

### Lancement du logiciel

* Ouvrez l’IDE de votre choix (ex : Spyder).
* Chargez le fichier Python « *IHM.py* ».
* Assurez-vous que les documents « *Tableau.png »*, « Logo\_*AndroMach.png », « Synoptique\_Allumeur.png »* se trouvent dans le même répertoire que « *IHM.py* ».
* *Rédiger les fichiers de séquence (au moins un) dans des .txt en suivant le format décrit précédemment. Assurez vous également qu’ils se trouvent dans le même répertoire que* « *IHM.py* »
* Connecter la carte STM à la Beagle Bone via un câble Ethernet et connecter
* Paramétrer les adresses de communication sur la STM, Beagle Bone et l’IHM
* Vérifiez la bonne transmission en utilisant la commande « ping » de l’invité de commande. *Pingez les adresses IP de la Beagle Bone et de la STM*
* Une fois ces modifications effectuées, exécutez le script « *IHM.py* ».
* Si l’interface apparaît correctement et qu’aucun message d’erreur n’est affiché dans le terminal de votre IDE, l’application est prête à usage.
* Si des problèmes persistent, il est probable que les adresses IP utilisées soient la source du problème.

### Interface utilisateur

Ci-dessous, un aperçu de l’interface utilisateur du logiciel :

A diagram of a machine

Description automatically generated

Figure III‑I : Interface utilisateur du logiciel de pilotage du banc d'essai

Cette interface est composée de 5 sections principales.

**Synoptique et contrôle des vannes :**

La section principale est le synoptique du banc de l’allumeur. Il résume les principaux éléments constituant la fluidique du banc tels que les vannes (solénoïde ou manuel), les réservoirs, les capteurs, l’allumeur et les différents tuyaux reliant ces éléments. Pour l’accompagner, 7 sous-sections sont présentes. Chacune d’entre elle permet de commander manuellement les électrovannes. Une sous-section est composée : d’une frame, d’un label qui rappelle le nom de la vanne, un label qui rappelle l’état actuel de la vanne (ouvert/fermé) et de deux boutons servant respectivement à ouvrir et fermer la vanne. L’état d’une vanne est affiché sur sa sous-section associée et dans un tableau récapitulatif. Ce tableau a pour but de résumer l’état de toutes les vannes évitant ainsi de chercher en permanence la sous-section d’une vanne pour voir son état. Le bouton « SPARK » permet à l’utilisateur d’allumer la bougie. Ce bouton change de couleur en fonction de son état. Un bouton rouge signifie une bougie éteinte et un bouton vert signifie une bougie allumée. Enfin, des labels sont visibles au niveau des différents capteurs pour afficher en temps réel leurs valeurs.

**Choix et lancement d’une séquence**

Sur la partie droite de l’IHM se trouve 4 zones, la première concerne le choix et le lancement d’une séquence d’allumage. Une combobox permet de répertorier les différents fichiers (.txt) que vous aurez rempli à l’avance. Ces fichiers doivent suivre un format bien précis.

A screen shot of a computer

Description automatically generated

Figure III‑II : Exemple de fichier de séquence

*Attention : Prenez garde au format. Ne remplacer pas les virgules, ne rajouter pas de ligne ou d’espace et assurer vous que le nombre de chiffre dans la première ligne correspond bien avec le nombre de vanne. Les temps doivent forcément être rangés dans l’ordre. Il est fortement conseiller de garantir un certain délai entre deux changements d’état.*

Il suffira de cliquer sur l’un d’eux pour le sélectionner. Le bouton « View checklist » permet d’afficher un chronogramme de la séquence. Enfin, le bouton « Start » permet d’entamer le processus d’allumage. En cliquant dessus, la STM va configurer les vannes dans un état initial et attend la confirmation de l’utilisateur pour continuer. Un chronomètre va alors s’afficher. Le premier bouton « Start Countdown » permet d’entamer le décompte avant allumage et le bouton « Stop » permet d’interrompre le décompte. Lorsque le décompte tombe à zéro, le contrôle des vannes sur le synoptique est bloquée et un chronomètre se lance pour afficher le temps depuis allumage.

A screenshot of a computer

Description automatically generatedA screenshot of a clock

Description automatically generated

Figure III‑III : Affichage du choix et du lancement d’un essai

**Affichage des graphiques**

La deuxième zone permet d’afficher les graphiques. Cette nouvelle fenêtre peut être décomposé en deux sections : les graphiques en temps réel et les options d’affichage. Les options d’affichage concernent la taille de l’axe « temps » c’est à dire le nombre de valeur à afficher à chaque instant. L’utilisateur a ainsi le choix d’afficher les 10 ou 30 dernières secondes, la dernière minute ou l’intégralité des valeurs depuis le lancement de l’IHM. Chaque grandeur (pression, température et débit) possède plusieurs capteurs impliquant donc plusieurs courbes. Par souci de lisibilité, il est possible de choisir les courbes à afficher. Il faut noter que cette fenêtre peut être ouverte ou fermée à n’importe quel moment même pendant le test.

**Emergency**

La troisième zone concerne l’arrêt d’urgence, mis en évidence par un bouton rouge. Il permet tout simplement d’interrompre toute action en cours que la carte STM pourrait être en train d’effectuer. Ce bouton est à utiliser principalement durant la phase de test si besoin. Une fois l’arrêt enclenché, une séquence est activée. Toutes les vannes sont, dans un premier temps, fermées puis une purge est effectuée en ouvrant les vannes nécessaires. Il est alors possible de recontrôler les vannes en cas de besoin. Ce bouton d’arrêt d’urgence interrompt également le chronomètre du test.  
  
**Fermeture de l’interface**

La dernière zone est le bouton fermeture. Afin d’éviter tout action involontaire, il a été décidé de désactiver le bouton de fermeture de fenêtre habituellement localisé en haut à droite de la fenêtre. Un nouveau bouton a été créé afin de le remplacer. Si celui-ci est sélectionné, une petite fenêtre warning apparaitra afin que l’utilisateur confirme la fermeture du programme.

## Exemple d’utilisation

Je souhaite mener un essai sur un allumeur torche. Après avoir vérifié les connectiques, le fonctionnement des différents éléments du banc et mis en place un périmètre de sécurité, je dois établir un ou plusieurs fichiers .txt au format adéquat ou vérifier le format de ceux déjà existant. Dans le programme « IHM.py », je dois vérifier les paramètres de lancement, notamment les adresses de communication et leur port et le type de nom du fichier csv (daté ou défaut). Je peux alors exécuter le script « *IHM.py* ».

L’interface graphique s’ouvre. Je vérifie rapidement la console de l’IDE pour s’assurer qu’aucun message d’erreur n’est présent. Je vérifie ensuite que les capteurs reçoivent des données et que celle-ci sont cohérentes. Il est également conseillé de vérifier le bon fonctionnement la fenêtre des graphiques notamment que l’affichage a bien lieu en temps réel.

En accord avec les procédures rédigées à l’avance, je teste l’ouverture et la fermeture des vannes.

*Attention : en cas de problème, ne fermer pas immédiatement l’interface car aucun reset de position n’est programmé. Il est donc possible qu’une vanne reste dans une position non désirée. Si toutefois, l’interface venait à fermer (volontairement ou non), l’interface est programmée pour forcer une position initiale des vannes a son lancement. Relancer le programme peut donc permettre un reset mais il reste préférable de ne pas recourir à cette solution.*

Une fois les premières vérifications effectuées et validées, je peux choisir la séquence voulu pour mon test. Je peux vérifier que la séquence correspond au test voulu grâce au bouton « View Checklist ». Je vérifie que les vannes sont positionnées correctement à partir du tableau récapitulatif. Je poursuis si aucune anomalie n’a été détecté. Lorsque le test est lancé, je peux ouvrir la fenêtre des graphiques pour observer les courbes et potentiellement détecté un problème. Si un important problème est détecté, fermer ou minimiser la fenêtre des graphiques et appuyer sur le bouton « EMERGENCY ». En accord avec les procédures et une fois la purge finie, ouvrir ou fermer des vannes.

Dans le cas où le test se déroule correctement, toujours en accord avec les procédures, réaliser les différentes actions sur les vannes (purge, vidange, …).

Pour relancer un essai, je ferme l’interface et la relance pour assurer l’initialisation des vannes et l’écriture des données dans un nouveau fichier csv.

Je peux également accéder au fichier de sauvegarde et appliqué les traitements et analyses nécessaires.

# Documents annexes

## Annexe A : Synoptique fluide du banc

A diagram of a machine

Description automatically generated

## Annexe B : Schéma de câblage

A diagram of a computer

Description automatically generated