TP de formation en Informatique

Objectif

Après avoir suivi des cours assez théoriques, il est temps de rentrer dans le vif du sujet et d'écrire du code pour le Club !!

Ce premier TP a pour objectif de te familiariser avec l'interface élec-info, pour gérer la communication via un bus UART.

Dans un premier temps, ton objectif est de contrôler une LED en l'allumant ou en l'éteignant.

Pour cela, voici un rappelle des couches d'abtractions du robot avec ce qui est déjà implémenté :

- **Stratégie** : rien là-dedans ne doit être fait (*ton rôle est de fournir une API bas niveau seulement, sans s'intégrer dans la stratégie du match.*)
- **Robot** : tu vas devoir ajouter la gestion de la LED dans un robot, et ce sera l'occasion pour toi de moduler ton propre robot
- **Modules** : le gros de ton travail se passe dans cette couche, tu vas devoir implémenter entièrement le module de la gestion d'une LED.
- **Communication** : Une implémentation est déjà proposée pour contrôler une unique LED, tu es libre de modifier ce code pour éventuellement complexifier les LEDs dessus (dans le cas où tu veux ajouter un afficheur 7-segments par exemple :P)
- **Électronique** : le code Rust est flashé sur la blackpill et est prêt à l'emploi, en considérant que le module LED a pour **ID 4**

1. Mise en place de l'environnement de travail

Avant tout, il faut faire quelques manipulations sur ton ordinateur pour mettre en place l'ensemble des outils.

Téléchargement du code source

Tu peux suivre les instructions de cette page pour télécharger le répertoire de l'informatique (https://github.com/ClubRobotInsat/info.git)

Ensuite, il te faut installer un certain nombre d'outils pour finaliser cette mise en place : utilisation du script d'installation.

Seule l'installation des outils et d'un compilateur récent sont requis ici.

Pour vérifier que tout marche, je t'invite à lancer ces commandes dans un temrinal :

```
cd <répertoire club robot>/info
git checkout simu-elec
```

```
mkdir -p build
cd build
cmake ..
make all -j 4
```

Pour être sûr que le code s'exécute comme il faut, tu peux lancer des tests automatisés :

```
ctest
```

ou encore

```
./test/unit_testing_all
```

Si tous les tests passent, félicitations tu vas bientôt pouvoir écrire du code à ton tour !!

CLion

Si tu ne l'as pas encore fait, je t'invite à installer **CLion** qui est un IDE de C++. Cet outil te permets de coder bien plus efficacement dans ce langage.

Ensuite, tu peux te rendre sur la documentation de la configuration pour set up CLion spécialement pour la partie info.

Tu peux à présent appuyer sur le bouton `Build all` pour t'assurer que tout marche depuis l'IDE.

Écriture du module pour la gestion d'une LED

Pour rappel, ce module doit servir d'interface entre la communication avec l'élec et pour une utilisation informatique plus haut niveau, dans laquelle on pourrait appeler la fonction

```
void set_state(bool on_off)
```

Pour cela, il faut te rendre dans le dossier `info/src/robot/Modules`. Tu peux y trouver des exemples de modules, notamment pour gérer l'IO et les servo moteurs.

Ton but est de t'inspirer de ces exemples pour écrire ton module de gestion d'une LED.

Afin de t'aider à mieux lire le code, voici des commentaires sur le module IO :

- ***IO2019.h*** : fichier header qui contient les déclarations du module IO.

```
/// Tous les fichiers header doivent commencer par ça, pour éviter des problèmes
#ifndef ROOT_MODULEIO2019_H
#define ROOT_MODULEIO2019_H

/// Inclusion de tous les fichiers dont on a besoin.
/// Pour inclure des librairies système, il faut utiliser '#include <lib_system>'
#include "Module.hpp"
```

```
/// Un 'namespace' permets de regrouper tout un tas de classes dans un même conte
/// Par exemple, toute la librairie standard est contenue dans le namespace 'std'
namespace PhysicalRobot {

    /// Définition du module IO, qui hérite directement d'un 'Module' (qui est ur
    /// Le 'Module' demande une structure C pour s'initialiser ; le code s'appuie
    class IO2019 final : public Module<SharedIO2019> {
    public:
        /// Constructeur du module depuis un id (de 0 à 15)
        explicit IO2019(uint8_t id);

        /// Getter pour accéder à l'état du module ; cette fonction est spéficiqu
        bool read_tirette() const;

        /// Retourne la taille d'une trame ; la fonction est virtuelle pure dans
        uint8_t get_frame_size() const override;

    private:
        /// Depuis le module IO, il faut pouvoir construire une structure C et la
        /// Petite question au passage, pourquoi ces deux fonctions sont-elles dé
        SharedIO2019 generate_shared() const override;
        void message_processing(const SharedIO2019&) override;

        /// Cette fonction n'est là que pour les modules qui agissent avec des ac
        void deactivation() override {}

        /// Variables qui définissent l'état du module IO
        bool _tirette;
    };
} // namespace PhysicalRobot

/// Les fichiers header doivent commencer par des '#ifndef .. #define' et se fini
#endif // ROOT_MODULEIO2019_H
```

- ***IO2019.cpp*** : fichier source qui définit les fonctions du module IO

```
/// Pour pouvoir définir ce que font les fonctions du module, il faut inclure leu
#include "IO2019.h"

/// Toutes les fonctions de la classe IO2019 sont dans le namespace 'PhysicalRobo
namespace PhysicalRobot {
    /// Constructeur du module. En appelant le constructeur du 'Module', de nombr
    /// Il faut juste donner les fonctions de parsing C pour la communication ; p
    IO2019::IO2019(uint8_t id) : Module(id, io_read_frame, io_write_frame) {}

    /// Cette fonction permets de demander à l'électronique de lire un capteur po
    bool IO2019::read_tirette() const {
        /// Cette ligne obscure permets une exécution asynchrone : pour avoir acc
        /// Pour les LEDs, tu peux aussi appeler les macros 'lock_variables()' et
        std::lock_guard<std::mutex> lk(_mutex_variables);
        return _tirette;
    }

    /// Le code C retourne directement la taille occupée par la trame. L'appel C
    uint8_t IO2019::get_frame_size() const {
        return get_size_io_frame();
    }
```

```
    /// Construction de la structure C depuis les variables de la classe
    SharedIO2019 IO2019::generate_shared() const {
        SharedIO2019 s = {static_cast<uint8_t>(_tirette), 0};

        return s;
    }

    /// Mise à jour du module depuis le code électronique
    void IO2019::message_processing(const SharedIO2019& s) {
        std::lock_guard<std::mutex> lk(_mutex_variables);
        if(s.parsing_failed == 0) {
            _tirette = static_cast<bool>(s.tirette);
        }
    }
} // namespace PhysicalRobot
```

À présent, à toi de jouer ! Il faut :

- créer une nouvelle classe
- ajouter le fichier `.cpp` dans le `CMakeLists.txt` correspondant (CLion va t'aider, et le fichier est explicite ; il faut bien modifier le `set MODULE_SOURCES`)
- Inclure ton nouveau module dans le fichier `ModuleManager.h`

Utilisation du module LED pour créer un robot

Tu vas bientôt pouvoir voir le fruit de tes efforts !

Mais avant tout, il faut définir une instance de `Robot` pour avoir accès à la communication UART automatisée. N'oublie pas, le module IO doit avoir l'ID 4 pour que la communication avec l'élec se passe bien *(oui, ce choix est totalement arbitraire)*.

Tu peux te simplifier la tâche en allant voir la construction d'un robot qui possède un manager de servo-moteurs dans le fichier `info/test/servos.cpp`.

Soit tu modifies directement ce fichier, soit *(mieux)* tu crées ton propre fichier `led.cpp` dans lequel il faut créer un manager de modules, un robot puis se connecter avec l'électronique par une liaison série `RS232` sur la connexion `/dev/ttyACM0`.

À toi de faire jouer ton imagination pour jouer avec la LED : tu peux l'allumer / éteindre toutes les secondes, convertir un message en Morse pour faire clignoter la LED après une entrée utilisateur, ...

Set-up de la connexion

Si tu lances le programme, tu devrais voir une erreur qui stipule que la connexion à la connexion série a echoué : c'est normal vu que tu n'es pas branché à la carte élec !

Cependant, un simple utilisateur n'a pas les droits pour parler dessus. Pour configurer cette autorisation, il faut te rajouter au groupe `dialout` (ça ne sera effectif qu'au prochain login) :

```
sudo groupadd dialout
sudo usermod -a -G dialout <username>
```

Tu peux aussi lancer le programme de connexion en `sudo` à chaque fois, ça marche aussi.

Branchement électronique

Si tout a bien été configuré, la carte blackpill que tu as à disposition est déjà flashée avec un code qui allume la LED de la carte si elle reçoit un `1`, et qui éteint la carte à la réception d'un `0`.

Dans un premier temps, branche le câble rouge à un port USB de ton PC, si la carte s'allume de 1000 feux c'est qu'elle est bien alimentée.

Il faut ensuite appuyer sur le bouton noir pour la reset : on réinitialise son état pour qu'elle soit cohérente.

Enfin, tu peux lancer ton application pour allumer ou éteindre la LED en fonction de tes choix d'implémentation et tu devrais voir la LED clignoter : si ça marche ***BRAVO*** le TP est fini ! Tu peux complexifier le fonctionnement de la LED pour communiquer en Morse par exemple, ou alors lire les classes `Module`, `ModuleManager` et `Communicator` ainsi que le code C (qui est présent dans `info/elec/librobot/c_src`) pour avoir une meilleure idée du fonctionnement global.

Si ça ne marche pas, tu peux télécharger l'application du *Logic Analyzer* pour observer ce qui passe sur la connexion série *(c'est pour les utilisateurs avancés donc demande de l'aide aux anciens)*, vérifier scrupuleusement ton code, comparer avec celui écrit par tes collègues ou encore rajouter plein de logs pour savoir ce que fait ton programme ; bref connaître les joies du débug :D

Mot de la fin

Si tu es parvenu.e à la fin de ce tuto, félicitations car tu viens de prendre en main une partie non négligeable du code : celle qui permets à l'électronique et à l'informatique de communiquer.

Je t'invite bien entendu à lire ce que tu peux des fonctions autour du module LED, pour comprendre un maximum le lien entre tous ces fichiers.

J'espère que cette initiation au code informatique ne t'a pas donné trop de sueurs froides et que tu as compris un maximum de concepts !!