Guide d'utilisation

1. Dans un premier temps, installer l'environnement de simulation.
2. Lancer l'application et le simulateur graphique, l'ordre ne compte pas : il est possible de fermer/relancer les deux parties à tout moment.
3. *Et voilà*

Fonctionalités du simulateur graphique

écran (1) écran 320x240 pixels du M5Stack Core 2, prend en charge le texte arbitraire, la taille de police, effaçage de l'écran et le retour automatique à la ligne.

Boutons M5 (2) 4 boutons physique du M5Stack Core 2: * **R** : reset - réinitialise le programme * **A** : bouton A - fonctionnalité selon le mode * **B** : bouton B - changement de mode * **C** : bouton C - fonctionnalité selon le mode

Matériel Simulé (3) Contrôles granulaire sur le matériel, pratique pour déclencher des cas d'erreurs durs à reproduire en réel.

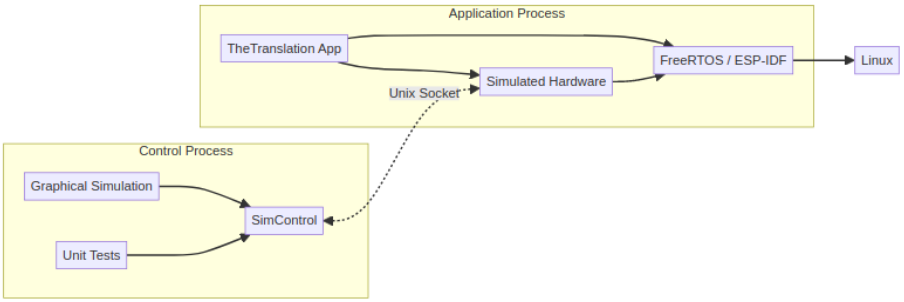
De haut en bas : 1. **Boutons M5** (voir ci-dessus) 2. ***Tag Reader*** - lecteur RFID des colis
* *Enable* - branche/débranche le composant * *Version* - version du firmware émulée, nombre hexadécimal. Seul les valeurs 88,90,91,92 ou 12 sont compatible avec l'application. * *UID* - code barre / identifiant de la balise NFC * *Send* - envoie l'UID renseigné à l'application 3. ***End of Line Reader*** - capteur RFID fin de course
Fonctionne pareil que le *Tag Reader*, fonction non-implémentée dans l'application (#22) 4. ***Conveyor*** - tapis roulant * *Enable* - branche/débranche le composant * *Speed* - vitesse actuelle du moteur, la valeur est contrôlée par l'application uniquement. 5. ***Sorter*** - servomoteur du tireur * *Enable* - branche/débranche le composant * *Angle* - angle du moteur, en degrés (?), la valeur est contrôlée par l'application uniquement. 6. ***WIFI Antenna*** - antenne WIFI * *Enable* - branche/débranche le composant * *Mode* - valeurs courantes (contrôlée par l'application uniquement) :
`WIFI_MODE_NULL`, mode non-déterminé `WIFI_MODE_STA`, mode connection "normale"
`WIFI_MODE_AP`, mode point d'accès, l'application expose un "serveur"
* *Status* - statut de la connection, valeurs courantes :
`WL_IDLE_STATUS`, inactif
`WL_NO_SSID_AVAIL`, connection impossible le SSID n'existe pas
`WL_CONNECTED`, connection établie
`WL_CONNECT_FAILED`, connection échouée * *Expected SSID*, nom SSID attendu pour la connection, doit correspondre à la valeur de la configuration `ap_ssid` pour établir une connection * *Expected Pass*, mot de passe attendu pour la connection, doit correspondre à la valeur de la configuration `ap_password` pour établir une connection

Configuration (4) Configuration persistante de l'application, les valeurs sont sauvegardées dans le fichier `./simulation.properties`.
* ***Save to file*** - sauvegarde les valeurs courantes dans le fichier * ***Apply changes*** - Pousses les modifications vers l'application * **Champs de configuration** - valeurs pour chaque option utilisée par l'application, modifiables uniquement qu'en mode configuration

Architecture

Le simulateur est composé de deux parties majeures: l'*application simulée* et le *contrôleur*
Il s'agit de deux processus (pas des *threads*) Linux différents qui communiquent un s'envoyant des messages via un socket Unix.

Vue haut-niveau du système



Application simulée

Le code applicatif de *TheTranslation* est lancé sur le PC hôte grâce à l'environnement Linux pour FreeRTOS/ESP-IDF. L'environnement est intégré à PlatformIO grace à la plateforme faîte-maison MisterPeModder/pio-esp32-hosted. La couche matérièle/pilotes est remplacée par des imitations qui envoie et reçoit des messages au processus *contrôleur*.

Détail d'implémentation Afin d'implementer la fonction "Réinitialiser" du simulateur, l'*application simulée* est en fait composée de ***deux*** processus : ***watchdog*** et ***simulation***. Le processus *watchdog* à pour rôle de (re)-lancer le processus *simulation* à chaque fois qu'elle quite sans plantage (c.à.d. via la commande "Réinitialiser".

Contrôleur

Le processus *contrôleur* pilote la simulation de l'application via deux modes : le mode test et le mode graphique. * En mode test, chaque test "unitaire" choisit les conditions de la simulation et regarde les résultats. * En mode graphique,

on affiche une représentation du M5Stack et son écran ainsi qu'une interface de gestion qui permet de manipuler l'état du materiel simulé en direct.

Installation

Simulateur de base

Pré-requis

- Linux (x86-64)
- Git
- PlatformIO

Installer l'environnement PlatformIO

1. Ouvrir un terminal
2. Naviguer dans le dossier `~/.platformio/platforms/` (créer si pas présent)
3. Cloner le dépôt Git MisterPeModder/pio-esp32-hosted avec le nom `esp32-hosted`
4. Vérifier l'installation : la commande `pio platform list` devrait lister la platforme `esp32-hosted`

En commandes

```
mkdir -p ~/.platformio/platforms
cd ~/.platformio/platforms
git clone https://github.com/MisterPeModder/pio-esp32-hosted esp32-hosted
pio platform list
```

Compilation

- **En ligne de commande**: `pio run -e m5stack-simulated`
- **VSCode**: Barre du bas > "Environement PIO" > "m5stack-simulated" puis "Build" (en haut à droite)
- **CLion**: Changer l'environement en "m5stack-simulated" et cliquer "Build" (en haut à droite)

Lancement

- **En ligne de commande**: `pio run -e m5stack-simulated -t exec`
- **VSCode**: Extension PIO > Platform > "Execute the built program"
- **CLion**: Cliquer sur le triangle vert

Tests unitaires

Suivre les étapes pour le simulateur de base ci-dessus.

Lancement

- **En ligne de commande**: `pio test --environment testing -v` (le flag `-v` est optionnel)

Simulateur graphique

Suivre les étapes pour le simulateur de base.

Pré-requis

- CMake
- Compilateur C++11 (GCC/Clang)
- Raylib (installée en librairie système)
- Ninja (optionnel)

Compilation

- **En ligne de commande**:

```
cd PROJECT_DIRECTORY
cd ./simulator-gui
cmake -DCMAKE_BUILD_TYPE=Debug -G Ninja -S . -B cmake-build-debug
cmake --build cmake-build-debug --target simulator-gui -j$(nproc)
```

- **VSCode**:
  1. Installer l'extension CMake
  2. Ouvrir le dossier `simulator-gui` en tant que projet
  3. Cliquer sur "build"
- **CLion**:
  1. Ouvrir le dossier `simulator-gui` en tant que projet CMake
  2. Cliquer sur "build"

Lancement

- **En ligne de commande**: (après la compilation) `./cmake-build-debug/src/simulator-gui`
- **VSCode**: Panneau CMake > Launch/Lancer
- **CLion**: Cliquer sur le triangle vert