



Intro. Lego Mindstorm EV3

David SALLÉ (david.salle@ensemblescolaire-niort.com)

Ce document est mis à disposition selon les termes de la licence

[Creative Commons BY-NC-SA 3.0](https://creativecommons.org/licenses/by-nc-sa/3.0/)



Version du document : v0.1

Date : 14/09/2021

Table des matières

1 - Introduction.....	3
2 - Robotique.....	3
2.1 - Définition.....	3
2.2 - Composants.....	4
2.3 - Machine à états finis.....	5
3 - Robot Lego Mindstorm EV3.....	7
3.1 - Présentation.....	7
3.2 - Mise en œuvre du robot.....	8
3.2.1 - A faire en début de séance.....	9
3.2.2 - A faire en fin de séance.....	9
3.2.3 - A faire en cas de blocage.....	9
4 - Environnements de développement.....	10
4.1 - Environnement de développement « simple ».....	10
4.1.1 - Installation de la chaîne de développement croisé.....	10
4.1.2 - Outils.....	11
4.1.3 - Hello world from EV3.....	12
4.2 - Environnement de développement « Docker ».....	13
4.2.1 - Présentation.....	13
4.2.2 - Installation de Docker.....	14
4.2.3 - Installation du conteneur de compilation croisée.....	14
4.2.4 - Un premier « Hello world ».....	15
4.3 - Programmation du robot.....	16
4.3.1 - Présentation de l'API.....	16
4.3.2 - Compilation avec un Makefile.....	17
4.3.3 - Exemples.....	18

1 - Introduction

Ce document présente le robot Lego Mindstorm EV3 et sa mise en œuvre en langage C++ dans un environnement Linux.

Il se découpe en 4 parties :






- introduction de quelques grandes notions autour de la robotique
- présentation du robot, ses capteurs et actionneurs, son fonctionnement
- environnements de développement croisé (simple ou Docker)
- programmation en C++ du robot

2 - Robotique

2.1 - Définition

Un robot est un dispositif alliant **mécanique**, **électronique** et **informatique** accomplissant automatiquement des tâches dangereuses, pénibles, répétitives ou impossibles pour les humains.

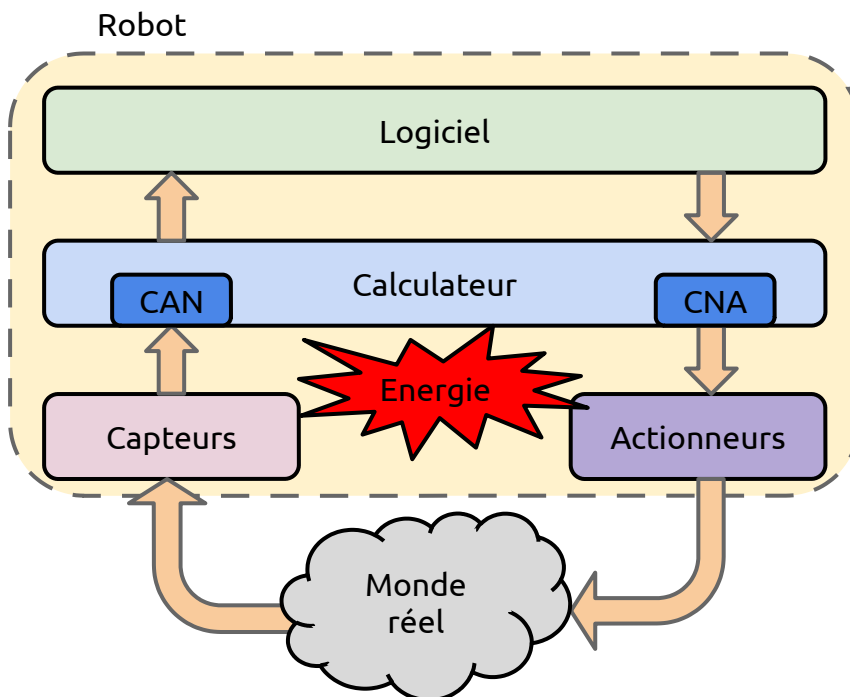
Exemples de robots et d'applications :

Robot aspirateur « Roomba »	Robot androïde « Enon »	Robot guerrier « Talon »	Robot industriel « Kawasaki »	Robot drone « Aeryon »
				

Source pour les images : Wikipédia

2.2 - Composants

Un robot est souvent composé des éléments suivants. Il peut être vu comme un ordinateur équipé de capteurs et actionneurs.



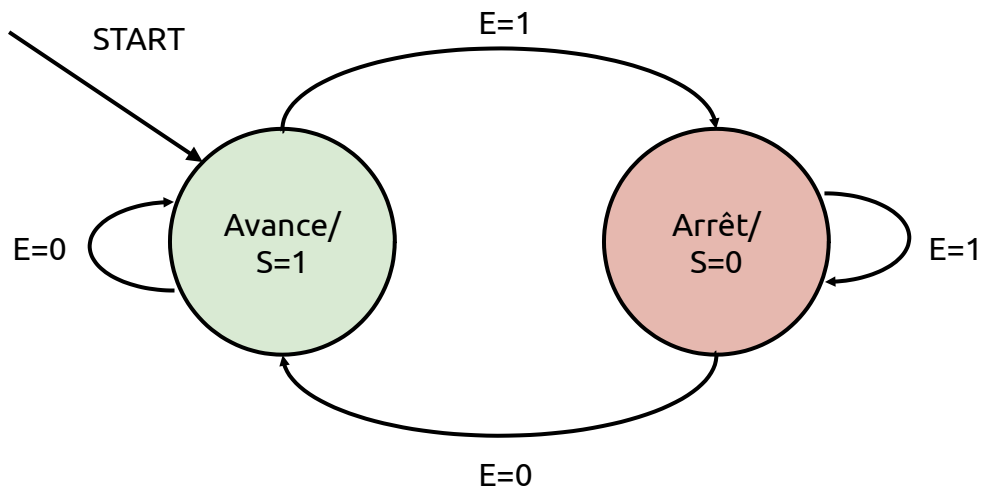
Tableaux des principaux composants d'un robot :

Composant	Description	Exemple
<i>Capteur</i>	Chargé de transformer une grandeur physique du monde réel en une tension électrique, puis en une information binaire à l'aide d'un CAN (convertisseur analogique/numérique)	Capteur de contact Capteur de distance Capteur de lumière GPS Gyroscope Accéléromètre Microphone
<i>Actionneur</i>	Chargé d'agir sur le monde réel en transformant une énergie en un phénomène physique utilisable. Un CNA (convertisseur numérique/analogique) permet de quantifier cette transformation	Moteur Vérin Écran Haut-parleur
<i>Calculateur</i>	Sorte de "carte mère" du robot	
<i>Logiciel</i>	Programme informatique pilotant le robot. Parfois utilisé au dessus d'un système d'exploitation.	

2.3 - Machine à états finis

La machine à états finis est utilisée pour modéliser les états et les actions d'un robot. Elle se représente comme suit et contient les éléments graphiques suivants :

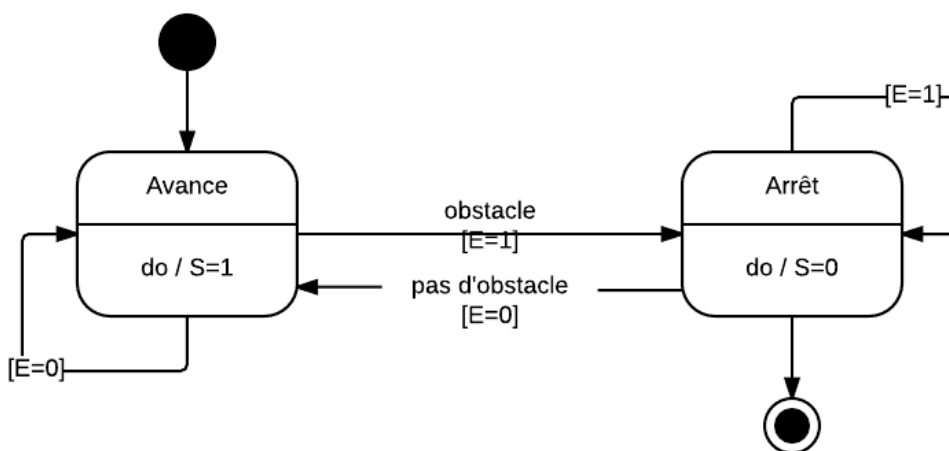
- les **états** : représentés par des ronds et associés à certaines actions qui pilotent les actionneurs du robot
- les **transitions** : représentées par les flèches entre les ronds et associées aux événements c'est à dire souvent aux capteurs du robots



Cette machine à états finis décrit un robot qui avance jusqu'à rencontrer un obstacle.

- E : représente le capteur/détecteur d'obstacle du robot
- S : représente l'actionneur/moteur qui fait avancer/arrêter le robot

Le diagramme états-transitions permet également de présenter une machine à états finis en utilisant le formalisme UML.



Cela pourrait se traduire en C++ comme suit :

```
// Librairies utilisées
#include "RobotExemple.hpp"

// Fonction main(), point d'entrée du programme
int main ()
{
    // Instanciation de l'objet robot
    RobotExemple robot;

    // Enumération des états possibles du robot + une variable
    enum EtatsDuRobot { AVANCE, ARRET };
    EtatsDuRobot etat = AVANCE;

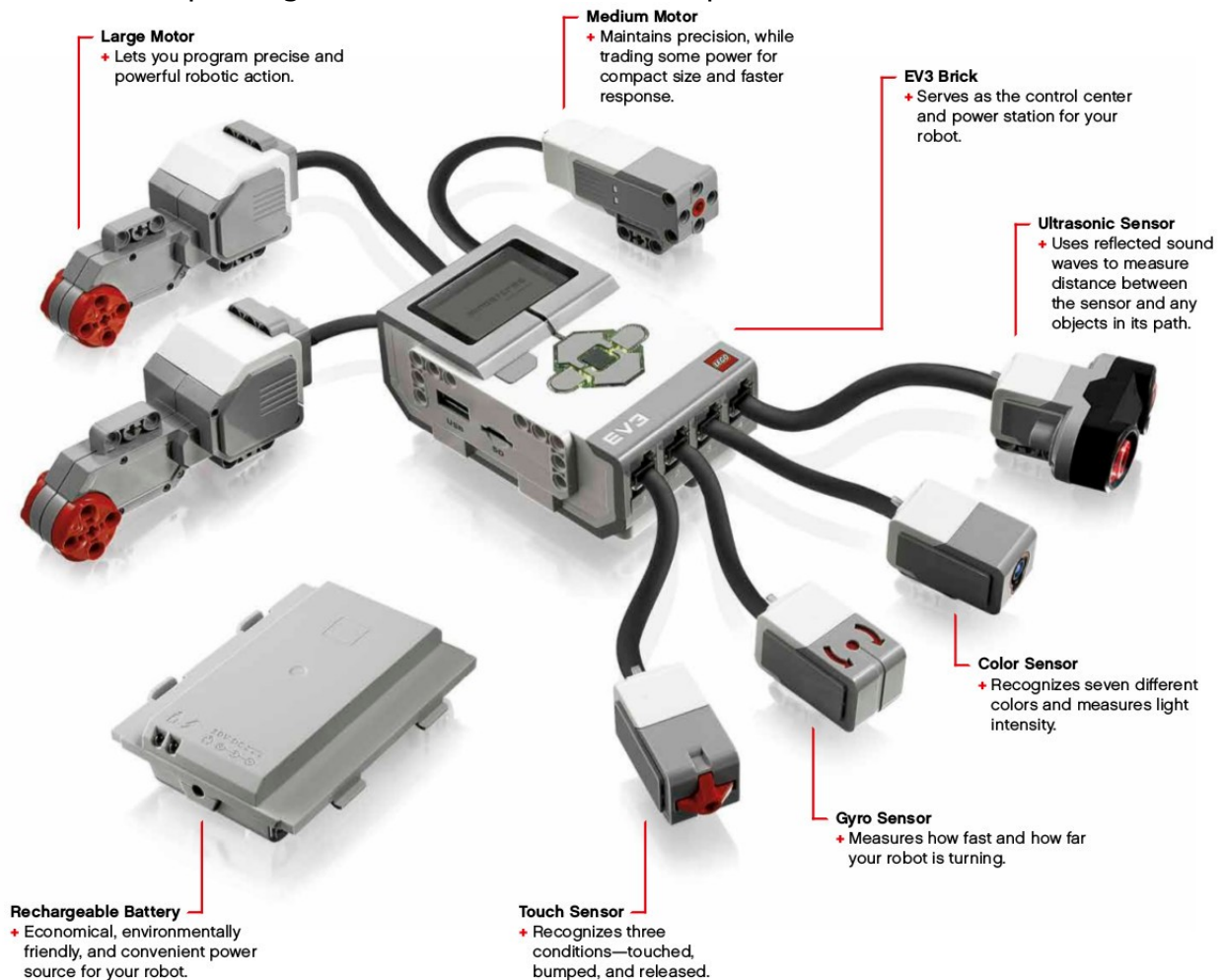
    // Boucle de gestion de la machine à états finis
    while (1)
    {
        // Etat AVANCE
        if (robot.getCapteur() == false)
        {
            etat = AVANCE;
            robot.avancer();
        }

        // Etat ARRET
        if (robot.getCapteur() == true)
        {
            etat = ARRET;
            robot.arreter();
        }
    }
}
```

3 - Robot Lego Mindstorm EV3

3.1 - Présentation

Le kit robotique Lego Mindstorm EV3 est composé des éléments suivants :



Caractéristiques de la brique :

Composant	Description
Processeur	ARM9 300MHz (Texas Instrument)
Mémoire	16Mo de flash + 64Mo de RAM + Slot µSDHC (max 32Go)
Communication	Bluetooth intégré + Wifi (avec dongle USB)
Écran	178×118 pixels en noir/blanc
Boutons/leds	5 boutons et 4 leds de couleurs (vert, rouge)
Son	Haut-parleur intégré

Caractéristiques des actionneurs :

Composant	Description
<i>Grand moteur</i>	vitesse = 160/170 tours/min couple = 20Ncm couple de blocage = 40Ncm capteur de rotation tachymétrique (1 pas = 1 degré) diamètre de la roue = 56 mm
<i>Moyen moteur</i>	vitesse = 240/250tours/min couple = 8Ncm couple blocage = 12Ncm capteur de rotation intégré (1 pas = 1 degré)

Caractéristiques des capteurs :

Composant	Description
<i>Contact</i>	Indique si enfoncé ou pas
<i>Lumière</i>	Mesure la couleur (noir=1, bleu=2, vert=3, jaune=4, rouge=5, blanc=6, marron=7 et inconnue=0) Mesure l'intensité de la lumière réfléchie (0 à 100), Mesure les composante RVB de la lumière (0 à 1020)
<i>Distance</i>	A ultra sons. Mesure de 3 à 250cm, précision +/- 1cm
<i>Gyroscope</i>	Mesure l'angle de rotation. Précision de +/- 3 degrés. Mesure aussi la vitesse angulaire (max 440 degrés/seconde)

3.2 - Mise en œuvre du robot

Une distribution Linux Debian Stretch **ev3dev** a été pré-installée et pré-configurée sur la carte microSD. Ce système d'exploitation prend le pas au démarrage sur le firmware officiel Lego.

<https://www.ev3dev.org/>

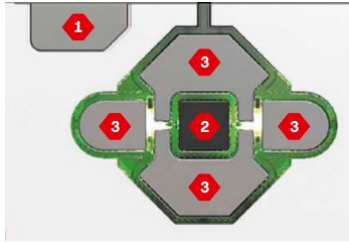
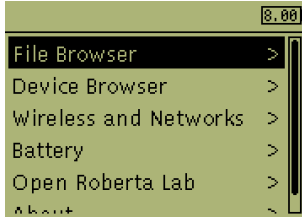


Les bibliothèques C++ permettant de piloter le robot ont également été pré-installées sur le robot. Reste l'environnement de développement croisé C++ à installer côté PC.

IMPORTANT : ces robots sont utilisés par différentes classes, chacune ayant un dossier dédié, faire attention donc au dossier utilisé :

- STS SNIR1 : **/home/robot/snir1**
- STS SNIR2 : **/home/robot/snir2**
- Terminale NSI : **/home/robot/nsi**

3.2.1 - A faire en début de séance

Manipulations pour mettre en œuvre le robot la première fois.

Action	Screenshot
Allumer le robot en appuyant pendant 2s sur le bouton central [2]	
Attendre le chargement du système d'exploitation Linux jusqu'à obtenir cet écran	
Le robot se connecte normalement au routeur Wifi de la salle D1 (SNIRAREA) et affiche son adresse IP obtenue via DHCP à la fin du processus. Noter alors l'adresse IP du robot 192.168.1.X	
Connecter le PC au robot en lançant un terminal et en y tapant la commande suivante : \$ ssh robot@192.168.1.X Le mot de passe est " maker " <i>//! à la première connexion, il vous demandera de créer une clef de chiffrement, répondre « yes » !/</i>	

3.2.2 - A faire en fin de séance

En fin de séance, il faudra :

1. sauvegarder ses travaux
2. appuyer plusieurs fois sur le bouton [1] en haut à gauche jusqu'à voir apparaître le menu avec l'entrée « **Power off** » qu'il faudra alors sélectionner.

3.2.3 - A faire en cas de blocage

En cas de blocage irrémédiable, il est possible de **rebooter** le robot en appuyant simultanément sur les boutons [1]+[2]

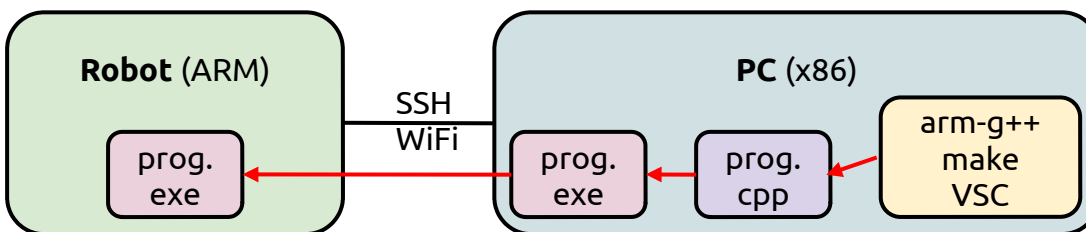
4 - Environnements de développement

Dans la suite de ce chapitre, 2 environnements vous sont présentés :

- version « **simple** » pour des programmes mono-thread
- version « **complexe** » avec un conteneur Docker, pour du multi-threads

4.1 - Environnement de développement « simple »

L'environnement de développement utilisé pour programmer le robot utilisera une chaîne de développement croisé sous Linux.



En effet l'architecture du robot (ARM) est différente de celle sur le PC (x86). Comme le robot n'est pas assez puissant pour pouvoir héberger une suite de développement (compilateur, IDE, débogueur...), Il faut installer celle-ci côté PC. Voici les étapes utilisées en développement croisé :

1. édition du code source sur le PC
2. compilation sur le PC avec un compilateur spécifique pour l'architecture du robot
3. transfert du programme exécutable du PC au robot
4. exécution du programme sur le robot

4.1.1 - Installation de la chaîne de développement croisé

Ouvrir une console sur le **PC** de développement. Télécharger et installer le compilateur pour processeur ARM :

```
$ sudo apt-get install g++-arm-linux-gnueabi
```

Télécharger et installer l'outil des gestion des mots de passe SSH :

```
$ sudo apt-get install sshpass
```

Télécharger depuis Moodle l'archive **code_source_lego_ev3.zip** et décompresser l'archive

```
$ wget ...
```

Fermer la console ouverte au début de l'installation.

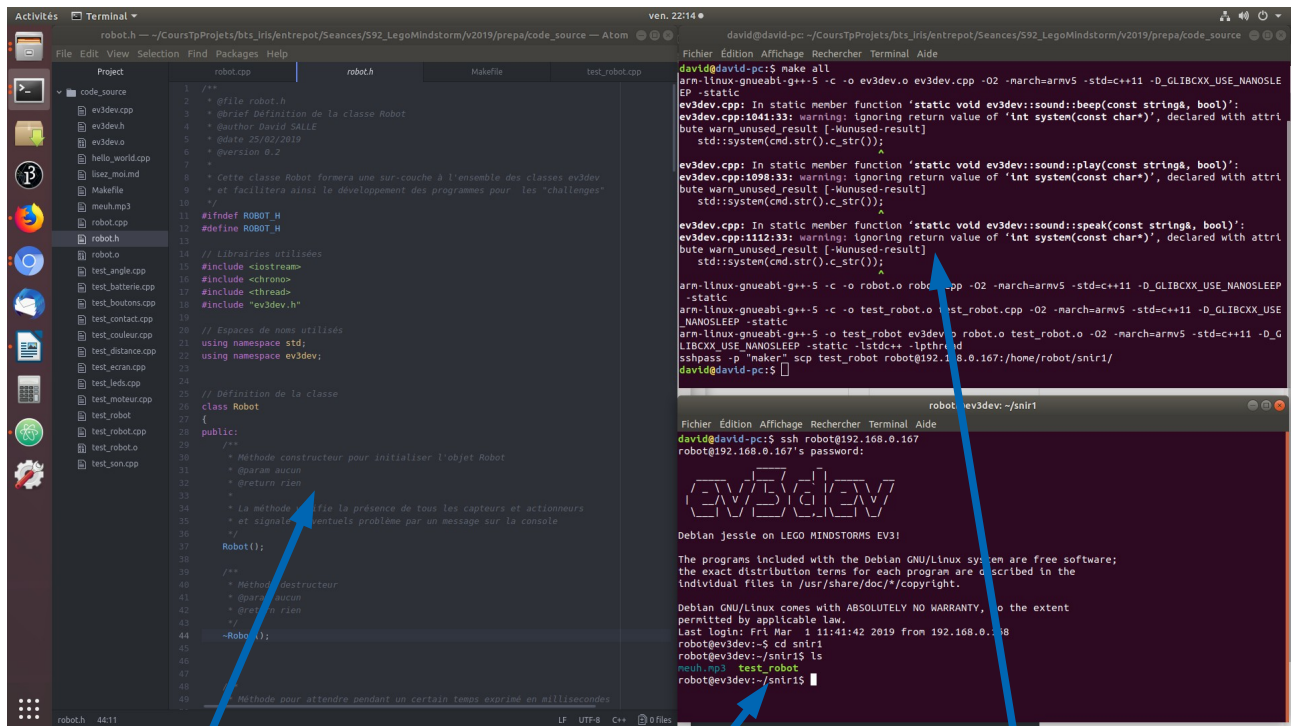
4.1.2 - Outils

Ci-dessous une possibilité pour développer votre projet Lego en utilisant :

- **Visual Studio Code/Atom/Geany** : comme éditeur de code source
- **Terminal_PC** : pour compiler et téléverser le programme
- **Terminal_ROBOT** : pour lancer les programmes via SSH

A noter qu'il est possible de les lancer un programme directement depuis le robot en utilisant les boutons :

- « **File Browser** » → « **snir1** » → « **mon_programme** »



VSC :
Édition du
code source

Terminal_ROBOT:
Communication SSH

Terminal_PC :
compilation &
téléversement

A faire pour obtenir cet environnement de développement :

1. Lancer **VSC**. Menu « **File** », puis « **Open folder...** » et pointer sur le dossier **code_source_lego_ev3** décompressé au 3.3.1
2. Lancer un premier terminal (**Terminal_PC**) depuis le dossier **code_source_lego_ev3** avec clic droit « **Ouvrir dans un terminal** »
3. Lancer un second terminal (**Terminal_ROBOT**) et saisir les commandes :

```
$ ssh robot@192.168.1.X  
$ cd /home/robot/snir1
```

4.1.3 - Hello world from EV3

Afin de tester et de valider l'installation de la chaîne de développement croisé et la liaison réseau avec le robot, vous allez éditer avec **VSC** le sempiternel « Hello world » disponible dans le fichier **hello_world.cpp** :

```
#include <iostream>

using namespace std;

int main()
{
    // Affichage
    cout << "Hello world from EV3 !!!" << endl;

    // Fin du programme
    return 0;
}
```

Dans le **Terminal_PC** entrer la commande permettant de compiler ce programme pour l'architecture ARM du robot.

```
$ arm-linux-gnueabi-g++ hello_world.cpp -o hello_world
```

Reste à transférer ce programme sur le robot avec la commande **scp** qui permet de copier des fichiers via une liaison SSH

```
$ scp ./hello_world robot@192.168.1.X:/home/robot/snir1/
```

Dans le **Terminal_ROBOT** entrer les commandes suivantes qui permettent de vous déplacer dans votre dossier de travail et d'exécuter le programme :

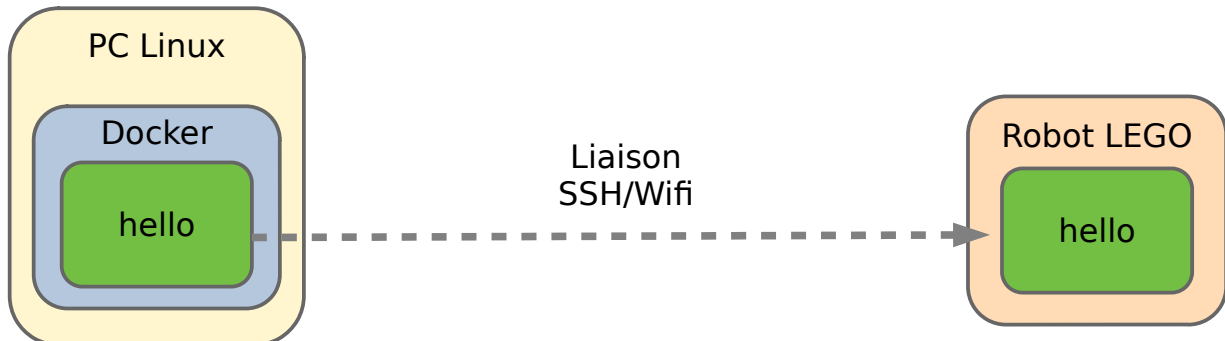
```
$ cd /home/robot/snir1
$ ./hello_world
```

Vous devriez alors voir apparaître le message dans la console, si tout s'est bien déroulé.

4.2 - Environnement de développement « Docker »

4.2.1 - Présentation

Ci-dessous le synoptique de l'environnement de développement basé sur Docker



L'environnement de travail côté PC repose sur un conteneur Docker qui permettra de faire du développement croisé avec le robot. Le conteneur embarque un compilateur et toutes les librairies nécessaires.

L'intérêt d'utiliser un conteneur Docker ici, est qu'il permet d'éditer facilement et de compiler rapidement nos programmes par rapport à une compilation in situ sur le robot. De plus les librairies sont les mêmes que sur le robot ce qui permet de faire du développement multi-threads.

Afin de bien distinguer la console Linux, la console Docker et la console Robot, le code couleur suivant sera utilisé :

```
pc:$ ls
docker:$ ls
robot:$ ls
```

Les fenêtres : IDE, console docker et console robot

```
pc:$ ls
docker:$ ls
robot:$ ls
```

```
hello_world.cpp - code_source_lego_ev3 - Visual Studio Code
1  /**
2   * @file hello_world.cpp
3   * @brief Le classique des classiques
4   * @author David SALLE
5   * @date 25/02/2019
6   * @version 0.2
7   *
8   * Affiche un message dans la console.
9   * Il sert surtout à valider l'installation de la chaîne de développement.
10  */
11
12  #include <iostream>
13
14  using namespace std;
15
16  int main()
17  {
18      // Affichage
19      cout << "Hello world from EV3 !!!" << endl;
20
21      // Fin du programme
22      return 0;
23  }
24
```

```
compiler@5e96f77e8ef9:/src$ ls
Dockerfile  demo_couleur.cpp  demo_moteur  ev3dev.o  test
Makefile    demo_distance.cpp demo_moteur.cpp  hello_world  test_threads_posix
demo_angle.cpp  demo_ecran.cpp  demo_son.cpp  robot.cpp  test_threads_posix.cpp
demo_batterie.cpp  demo_leds  demo_tachy.cpp  robot.hpp
demo_boutons.cpp  demo_leds.o  ev3dev.o  robot.o
demo_contact.cpp  demo_leds.o  ev3dev.h
compiler@5e96f77e8ef9:/src$ make clean && make all
rm -f *.o
arm-linux-gnueabi-g++ -c -o ev3dev.o ev3dev.cpp -O2 -march=armv5 -std=c++11 -D_GLIBCXX_USE_NANOSLEEP
arm-linux-gnueabi-g++ -c -o robot.o robot.cpp -O2 -march=armv5 -std=c++11 -D_GLIBCXX_USE_NANOSLEEP
arm-linux-gnueabi-g++ -c -o demo_leds.o demo_leds.cpp -O2 -march=armv5 -std=c++11 -D_GLIBCXX_USE_NANO
SLEEP
arm-linux-gnueabi-g++ -o demo_leds ev3dev.o robot.o demo_leds.o -O2 -march=armv5 -std=c++11 -D_GLIBC
X_USE_NANOSLEEP -lstdc++ -lpthread
sshpass -p "maker" scp demo_leds robot@192.168.0.184:/home/robot/snir2/
compiler@5e96f77e8ef9:/src$
```

```
robot@ev3dev:~/snir2$
bash: /usr/share/virtualenvwrapper/virtualenvwrapper_lazy.sh: Aucun fichier ou dossier de ce type
david@david-pc:~$ ssh robot@192.168.0.184
Password:
Linux ev3dev 4.14.117-ev3dev-2.3.5-ev3 #1 PREEMPT Sat Mar 7 12:54:39 CST 2020 armv5tejl

Debian stretch on LEGO MINDSTORMS EV3!
Last login: Mon Sep 13 13:46:51 2021 from 192.168.0.170
robot@ev3dev:~$ cd snir2
robot@ev3dev:~/snir2$ ./hello_world
Hello world from EV3 !!!
robot@ev3dev:~/snir2$
```

4.2.2 - Installation de Docker

L'installation de Docker est décrite sur le site officiel : <https://docs.docker.com/engine/install/>

4.2.3 - Installation du conteneur de compilation croisée

Les principales étapes après l'installation de Docker sont résumées ci-après. Pour de plus amples informations suivre le lien ici :

<https://www.ev3dev.org/docs/tutorials/using-docker-to-cross-compile/>

Créez un fichier **Dockerfile** avec le contenu suivant :

```
# Image de base pour notre conteneur
FROM ev3dev/debian-stretch-cross

# Installation des outils annexes
RUN sudo apt update
RUN sudo apt install -y sshpass
```

Construisez alors le conteneur avec la commande suivante. Le conteneur fait ~800Mo, donc prévoir un peu de temps et une bonne connexion !

```
pc:$ sudo docker build -t ev3env .
```

Sur Moodle, vous pourrez récupérer et décompresser une archive ZIP contenant les code sources C++ : **code_source_ev3dev.zip**

Lancez le conteneur. Vous devriez obtenir une console interactive liée au conteneur :

```
pc:$ sudo docker run --rm -it -v
/home/jbegood/projet_snirium/code_source_ev3dev:/src -w /src
ev3env
docker:$
```

Le dossier **/home/jbegood/projet_snirium/code_source_ev3dev/** est alors partagé entre le PC Linux et le conteneur Docker. C'est ce qui vous permet à la fois de :

- éditer votre code source dans l'éditeur sur Linux
- compiler votre code source dans le conteneur Docker

4.2.4 - Un premier « Hello world »

/!\ Pour les instructions qui suivent, soyez vigilant à la console utilisée /!

Dans l'IDE Visual Studio Code sous Linux, créer et éditer un fichier **hello.cpp** dans le dossier **/home/jbegood/projet_snirium/code_source_ev3dev/** avec le code suivant :

```
#include <iostream>
using namespace std;

int main()
{
    // Affichage
    cout << "Hello world from EV3 !!!" << endl;

    // Fin du programme
    return 0;
}
```

Pour compiler votre programme, vous pourrez utiliser la commande suivante (dans la console Docker)

```
docker:$ arm-linux-gnueabi-g++ hello.cpp -o hello
```

Puis vous pourrez téléverser le programme compilé avec (dans console PC) :

```
docker:$ scp hello robot@192.168.1.100:/home/robot/snir2/
```

Il faut ensuite se connecter via SSH au robot (dans console PC)

```
pc:$ ssh robot@192.168.1.100
```

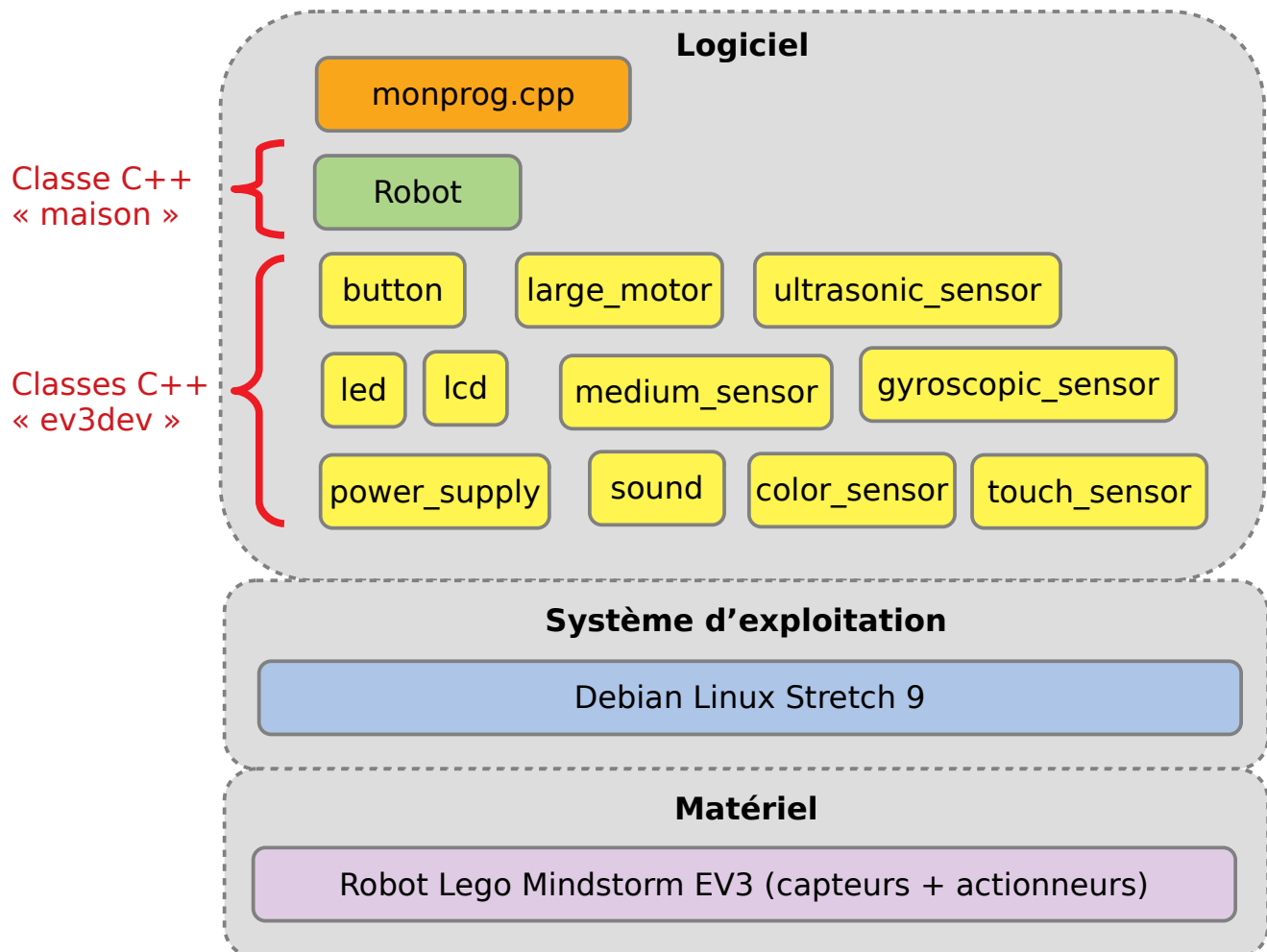
Pour exécuter le programme (dans console Robot)

```
robot:$ cd snir2
robot:$ ./hello
```

4.3 - Programmation du robot

4.3.1 - Présentation de l'API

L'API pour programmer le robot se résume à une classe **Robot**. Elle agit comme une interface en surcouches des classes ev3dev qui elles mêmes interagissent via le système d'exploitation avec le matériel (capteurs et actionneurs)



Votre projet sera constitué de 6 fichiers :

Fichier	Description	A modifier
monprog.cpp	Fichier contenant la fonction main()	oui
robot.hpp	Déclaration de la classe Robot	oui
robot.cpp	Implémentation de la classe Robot	oui
ev3dev.h	Déclaration des classes ev3dev	non
ev3dev.cpp	Implémentation des classes ev3dev	non
Makefile	Gestion de la compilation du projet	oui

Exemple de fichier **monprog.cpp** :

```
// Librairies utilisées
#include "robot.hpp"

// Point d'entrée du programme
int main()
{
    try
    {
        // Instanciation de l'objet robot
        Robot monRobot;

        // Appel des méthodes de la classe
        monRobot.faireQuelqueChose();
    }
    catch(ErreurRobot &erreur)
    {
        cout << erreur.recupererMessage() << endl;
    }

    // Fin du programme
    return 0;
}
```

4.3.2 - Compilation avec un Makefile

Le **Makefile** est un fichier qui permet d'automatiser la compilation d'un projet informatique, notamment lorsqu'il est composé de plusieurs fichiers.

Il est utilisé par la plupart des IDEs derrière le bouton de compilation.

Avant de compiler votre projet il faudra configurer le fichier **Makefile** avec **VSC**. Tout d'abord il faut adapter l'adresse IP du robot :

```
IP_ROBOT=192.168.1.X
```

Puis il faut préciser le nom du fichier contenant la fonction main(). Par exemple si vous travaillez avec un fichier **monprog.cpp**, il faudra modifier ainsi :

```
EXEC=monprog
```

Enfin pour **compiler** et **transférer** votre programme dans le robot, il suffira sur votre PC dans **Terminal_PC/Docker** de faire :

```
docker:$ make all
```

Le lancement se fera depuis **Terminal_ROBOT** comme au 3.3.3 ou via le « **File browser** » du robot.

4.3.3 - Exemples

Le tableau suivant présente les principales classes C++ de la librairie **ev3dev** illustrées par des codes sources disponibles sur Moodle. N'hésitez pas à tester les programmes, les modifier pour mieux comprendre le fonctionnement de l'API et du robot.

Composant	Fichier	Description
<i>Leds</i>	demo_leds.cpp	Allume/éteint les leds
<i>Boutons</i>	demo_boutons.cpp	Affiche l'état des boutons
<i>Buzzer</i>	demo_son.cpp	Parle, joue un son et un MP3
<i>Écran LCD</i>	demo_ecran.cpp	Allume/éteint un pixel de l'écran (Ne fonctionne pas)
<i>Batterie</i>	demo_batterie.cpp	Affiche l'état de la batterie
<i>Capteur contact</i>	demo_contact.cpp	Affiche l'état du capteur de contact
<i>Capteur distance</i>	demo_distance.cpp	Affiche la distance mesurée
<i>Capteur angle</i>	demo_angle.cpp	Affiche l'angle du robot
<i>Capteur couleur</i>	demo_couleur.cpp	Affiche la lumière (couleur, réfléchie et composantes RVB)
<i>Moteur</i>	demo_moteurs.cpp	Fait avancer le robot
<i>Tachymètre</i>	demo_tachy.cpp	Affiche la position du robot

Par exemple pour tester le programme **demo_leds.cpp** il faudra modifier le fichier **Makefile** avec **VSC** :

```
EXEC=demo_leds
```

puis compiler/téléverser le programme depuis **Terminal_PC/Docker** avec :

```
docker:$ make all
```

et enfin l'exécuter depuis **Terminal_ROBOT** avec :

```
robot:$ ./demo_leds
```