Mini-projet Robot snirium

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

Version du document : v0.4
Date : 14/09/2021

Table des matières

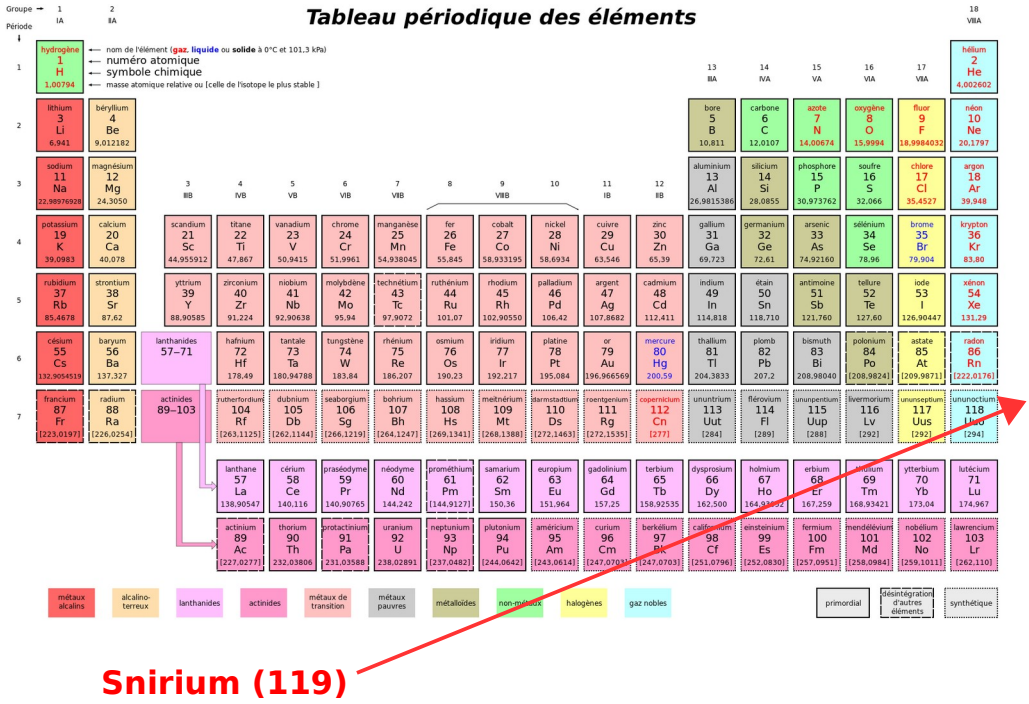
1 – Introduction

Ce mini-projet de synthèse doit permettre à des équipes de 2 étudiants de ré-exploiter les notions abordées en programmation système/réseaux, événementielle, les conteneurs Docker et les outils de développement.

2 – Cahier des charges

2.1 - Présentation

Une équipe de chercheurs de l'université de Balnave vient de découvrir un nouvel élément : le **snirium**. Dans la classification de Mendeleïev, il s'agit de l'élément 119, un métalloïde aux propriétés étonnantes puisqu'il permettrait de tripler les performances des panneaux photovoltaïques actuels !

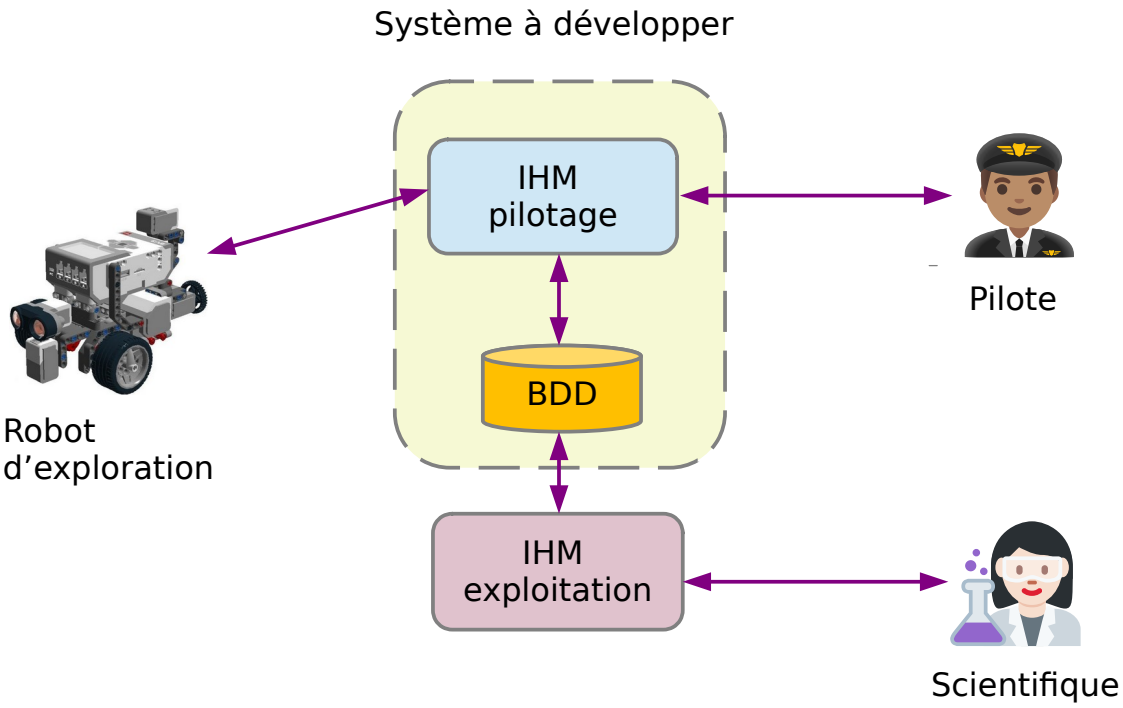


Ce métalloïde ne se trouve sur terre que dans les régions naturellement radioactive et donc dangereuses pour l'exploration. Plus précisément, on le trouve dans le sous-sol de grottes où une fumée épaisse empêche l'utilisation de caméra traditionnelle.

L'idée serait donc de piloter à distance un **robot d'exploration** équipé d'un capteur de snirium pour cartographier une zone donnée. Il serait équipé d'un capteur ultra-son en remplacement d'une caméra pour visualiser son environnement comme ce que peuvent faire les chauves souris (les ultra-sons sont utilisables dans les épaisses fumées contrairement aux ondes lumineuses)

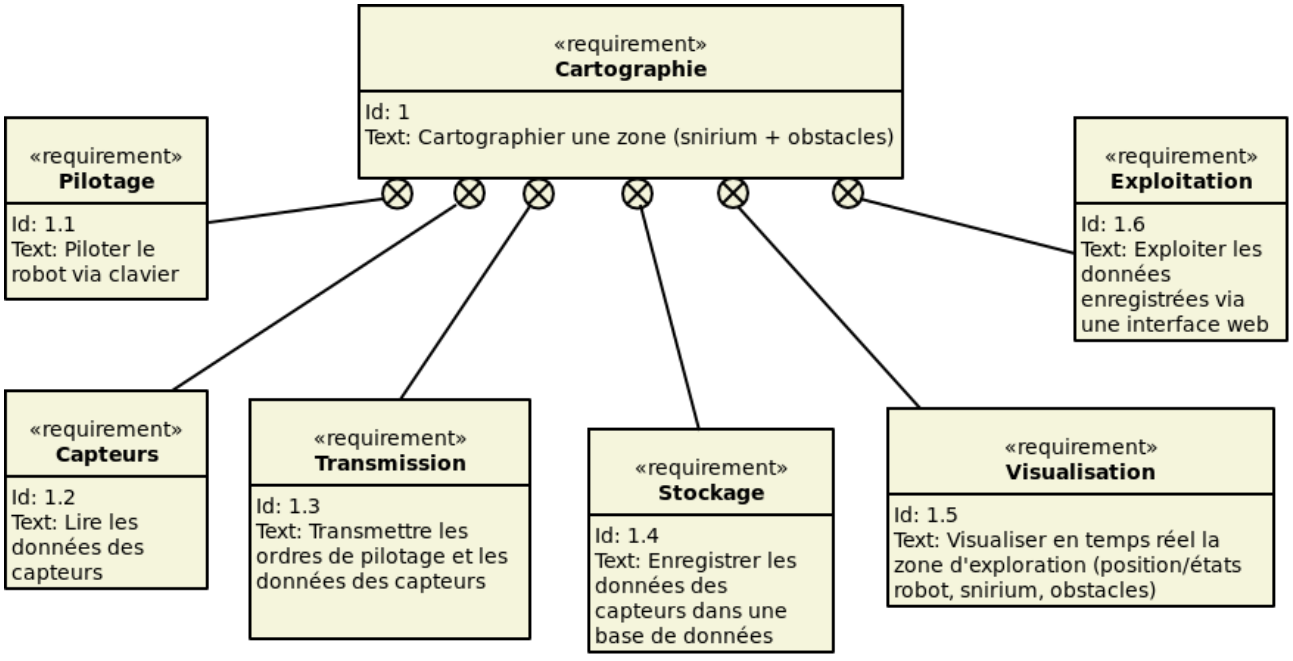
2.2 - Objectifs

Pouvoir **piloter** à distance un robot en évitant les obstacles et **récupérer** les mesures de présence de snirium effectuées sur le terrain pour les **enregistrer** dans une base de données.



2.3 – Expression du besoin

Ci-dessous le diagramme SysML des exigences. Chaque bloc exprime une fonctionnalité à réaliser.



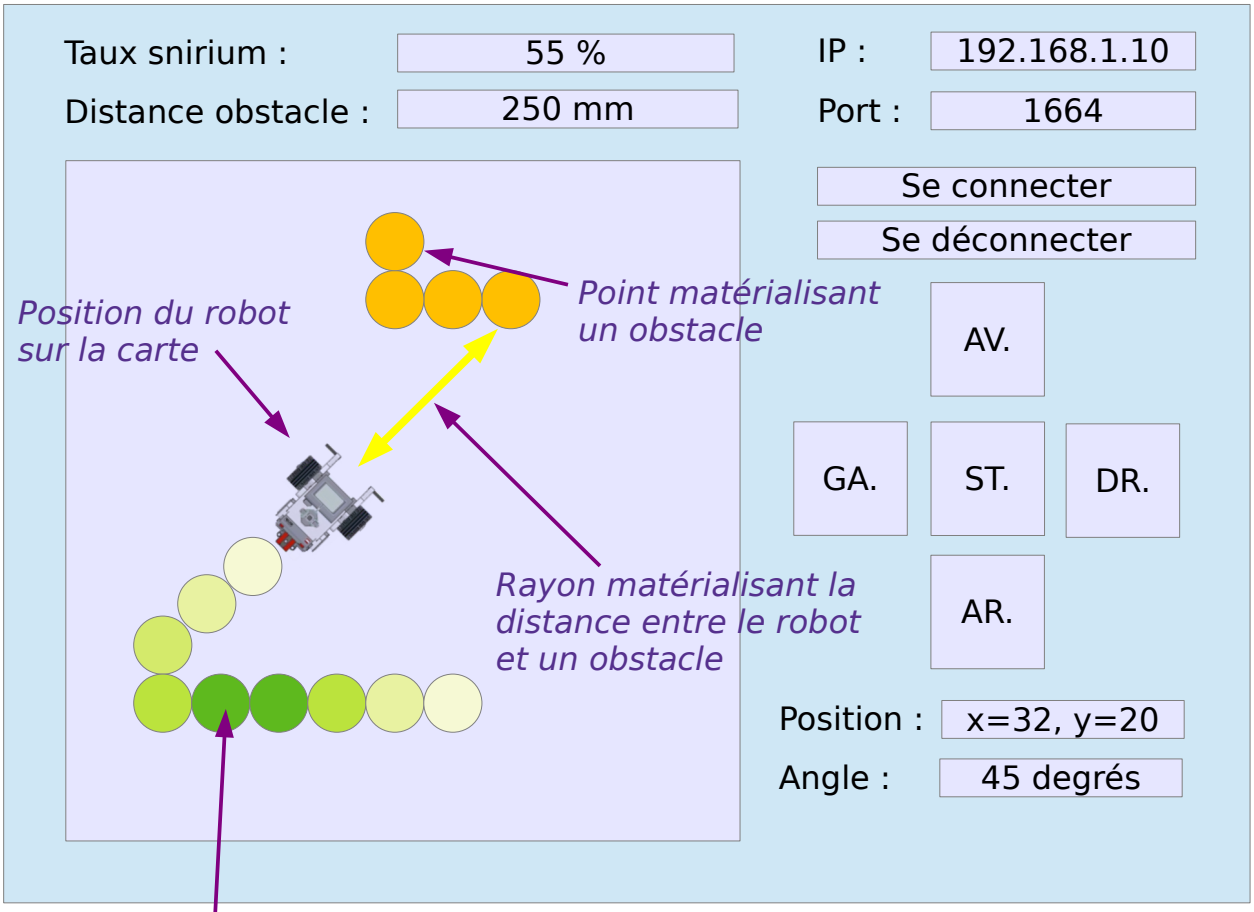
2.4 – Contraintes

Afin de ne pas surcharger le diagramme des exigences, les contraintes et précisions liées à chaque exigence sont rappelées dans le tableau ci-dessous.

Exigence	***Contraintes/précisions***
1.1 Pilotage	Le pilotage du robot se fera via le clavier d'un ordinateur. On retrouvera 6 ordres de pilotage associés à 6 touches • avancer • reculer • tourner à gauche • tourner à droite • lever barre • baisser barre
1.2 Capteurs	Le robot renverra au système différentes informations • le taux de snirium mesuré dans le sol (en pourcentage) • la distance robot/obstacle (en mm) • la position angulaire du robot (en degrés) • la position angulaire des 2 roues (en degrés)
1.3 Transmission	La transmission des ordres de pilotage et des informations se fera via une liaison réseau wifi. Idéalement cette liaison sera chiffrée pour éviter toute fuite d'informations Les données devront arriver toutes les secondes environ
1.4 Stockage	Le stockage des informations en provenance du robot se fera dans une base de données relationnelle afin de faciliter son exploitation ultérieure
1.5 Visualisation	Le pilote du robot pourra visualiser en temps réel (délai<1s) sur une IHM la zone d'exploration avec : • la position du robot (à calculer à partir des données des capteurs : xr, yr, angle) • les obstacles (xo, yo) • les points d'échantillons des taux de snirium (xs, ys, couleur) Voir le détail de l'IHM attendue au 2.5
1.6 Exploitation	Non demandée dans ce mini-projet

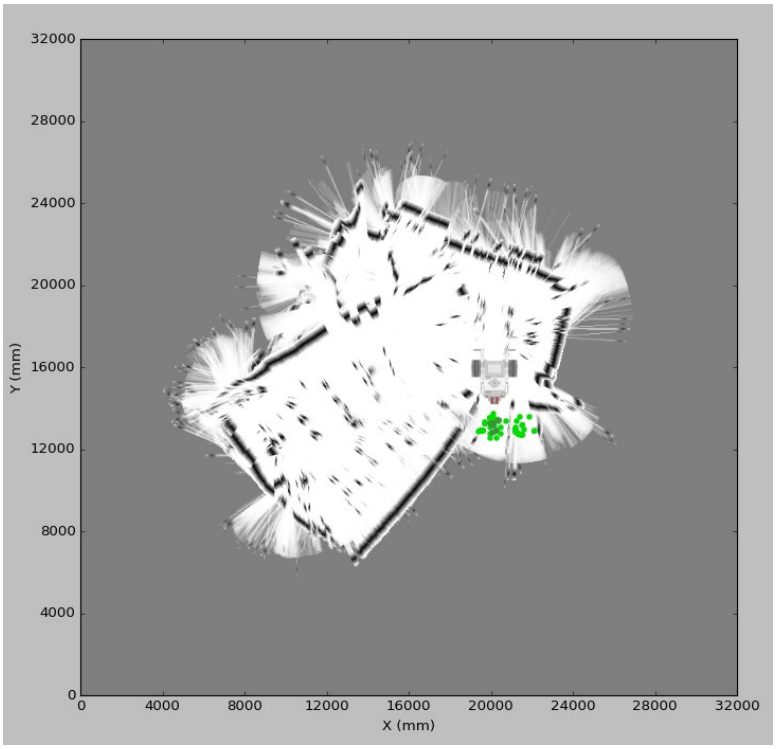
2.5 – IHM

Proposition d'IHM de pilotage :



Points de mesure Snirium : plus le taux est important plus la couleur est foncée

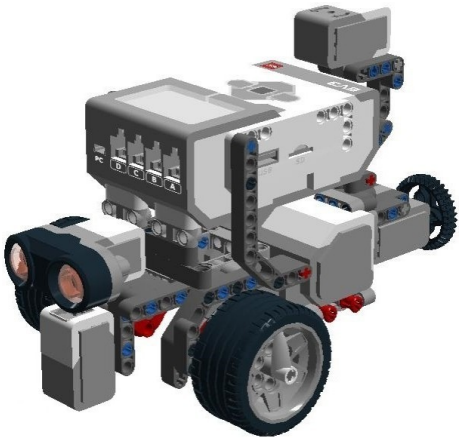
Résultat possible en dessinant chaque rayon se terminant par un point sombre pour matérialiser les obstacles



2.6 – Ressources

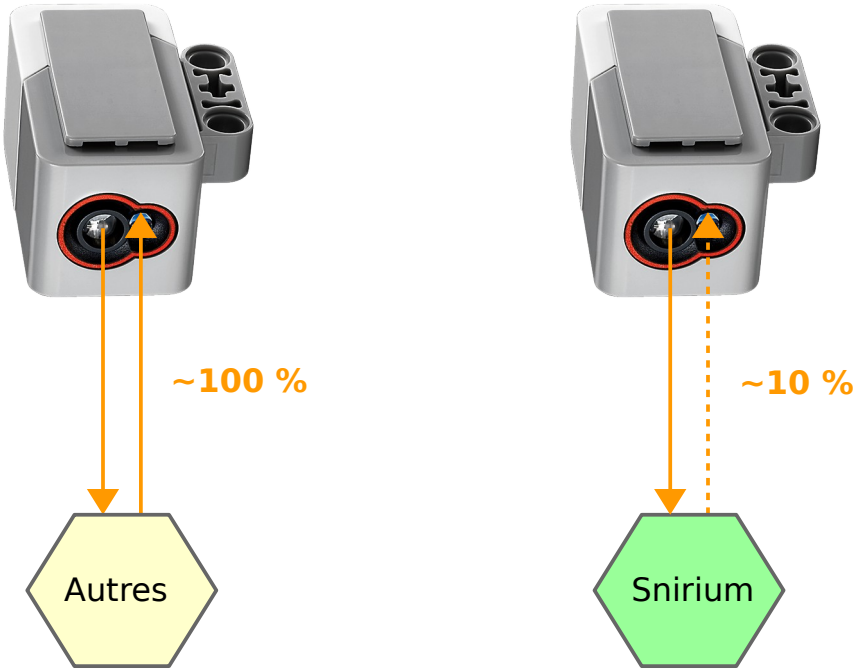
2.6.1 – Système d’exploitation

Le robot utilisé sera un **EV3** de chez LEGO équipé d'un système d'exploitation Linux Debian 9 (Stretch) et d'un dongle wifi afin de pouvoir le piloter à distance et récupérer les mesures effectuées.



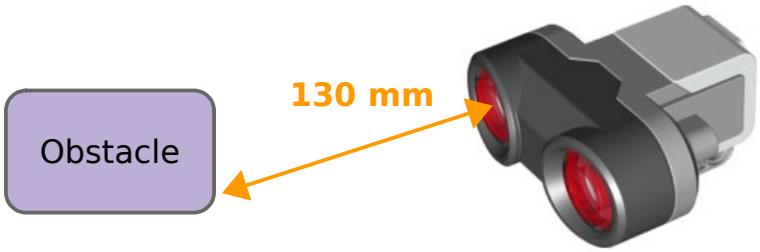
2.6.2 – Capteur de Snirium

Le robot portera un capteur de snirium. Ce capteur envoie un signal spécial dans le sol. Si celui-ci revient à 100 % c'est qu'il n'y a pas de snirium en dessous. Si le signal ne revient que faiblement par exemple à 10 %, c'est que du snirium se trouve en dessous et qu'il a absorbé une grande partie de l'énergie du signal d'origine.



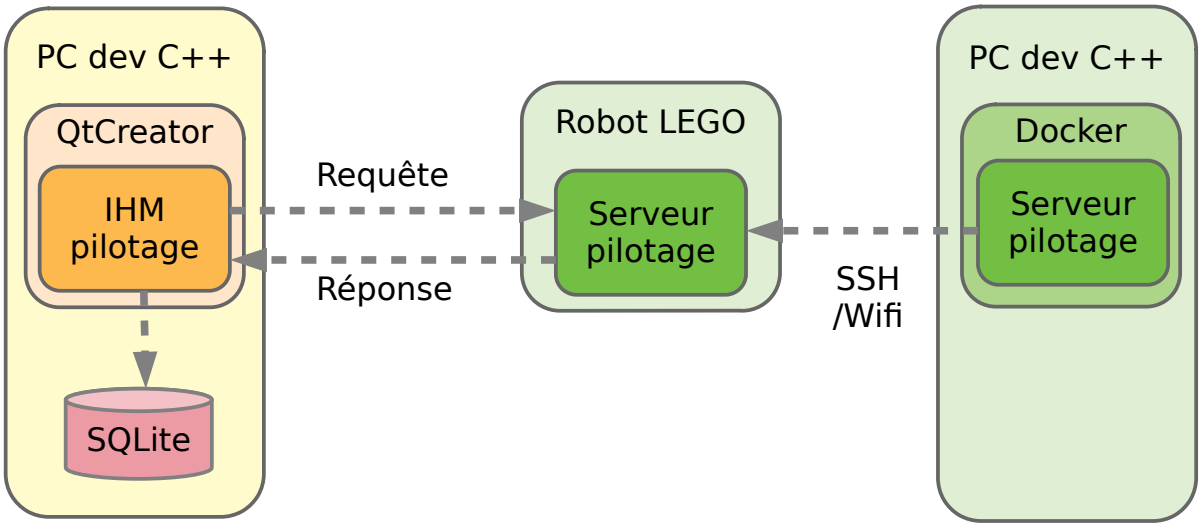
2.6.3 – Capteur ultrason

Le robot utilisera également un capteur ultrason. Ce dernier permettra de mesurer la distance entre d'éventuels obstacles et le robot. Le pilote à l'instar d'une chauve souris pourra ainsi éviter les obstacles présents sur la zone de recherche.



2.6.4 – Logiciels

Ci-dessous le synoptique global de votre environnement de développement



Le PC de développement du serveur de pilotage C++ utilisera :

- un environnement simulant le système d'exploitation, le compilateur et les librairies du robot dans un conteneur **Docker**
- un éditeur texte, par exemple **Visual Studio Code**
- l'outil **scp** et **sshpass** pour téléverser le programme dans le robot
- l'outil **ssh** pour exécuter le programme à distance sur le robot

Le PC de développement de l'IHM utilisera :

- L'IDE **QtCreator**
- Le gestionnaire de base de données **DB Browser for SQLite**

3 – Évaluation

3.1 – Travail à rendre

A la fin du mini-projet il vous faudra rendre par équipe, via Moodle :

- le code source C++ du programme serveur embarqué dans le « robot » dans une archive ZIP
- le code source Qt/C++ de l'« IHM » dans une autre archive ZIP
- le dossier technique au format PDF (décrire la solution, comment vous êtes arrivés au résultat)
    - *tâches, répartition*
    - *synoptique, schéma, screenshots*
    - *protocole réseau*

Parce que c'est un mini-projet complexe, vous devrez prioriser vos tâches dans votre travail.

Côté **robot**, on pourra ordonner les tâches comme ci-dessous :

1. un serveur TCP simple
2. un serveur TCP simple qui retourne de fausses valeurs
3. un serveur TCP simple qui retourne les vraies valeurs issues des capteurs
4. un serveur TCP simple qui reçoit et décode la requête de pilotage et retourne les vraies valeurs issues des capteurs
5. un serveur TCP simple qui reçoit et décode la requête de pilotage et exécute les ordres de mouvements et retourne les vraies valeurs issues des capteurs
6. un serveur TCP multithreadé qui…
7. un serveur TCP multithreadé implémentant le modèle producteur/consommateur avec des sémaphores qui…

Côté **IHM**, on pourra ordonner les tâches comme ci-dessous :

1. une IHM avec des éléments inactifs
2. une IHM avec des éléments qui effectue une requête au robot
3. une IHM avec des éléments qui effectue une requête au robot et reçoit/analyse la réponse (valeurs des capteurs)
4. une IHM avec des éléments qui effectue une requête au robot et reçoit/analyse la réponse (valeurs des capteurs) et met à jour les champs correspondants
5. une IHM avec des éléments qui effectue une requête au robot et reçoit/analyse la réponse (valeurs des capteurs) et met à jour les champs correspondants et calcule/dessine la position du robot, snirium, obstacle

3.2 – Critères d'évaluations

Critères d'évaluation :

	Exigence	--	-	0	+	++
Cahier des charges	1.1 : piloter le robot avec clavier					
	1.2 : lire données des capteurs					
	1.3 : transmettre ordre pilotage + données capteurs					
	1.3.1 : transmettre ordre de pilotage					
	1.3.2 : transmettre données capteurs + affichage					
	1.3.3 : chiffrer les transmissions					
	1.4 : enregistrer données dans bdd					
	1.5 : visualiser la zone d'exploration					
	1.5.1 : calculer/visualiser position robot (xr, yr)					
	1.5.2 : calculer/visualiser position obstacle (xo, yo)					
	1.5.3 : calculer/visualiser position snirium (xs, ys)					

	Items	--	-	0	+	++
Code	Commentaires					
	Indentation					
	Nommage des variables et fonctions					

	Items	--	-	0	+	++
Prog. Sys & rés.	Utilisation des sockets pour la mise en réseau					
	Utilisation des threads					
	Utilisation des sémaphores/mutex					
	Utilisation du modèle producteur/consommateur					
	Création d'un protocole d'échange des données					

5 – Éléments de codage

5.1 – Pour la communication

5.1.1 – Protocole

Les protocoles « texte » sont souvent les plus simples à mettre en œuvre. Le tableau ci-dessous situe le protocole SNIRIUM par rapport au modèle OSI :

Couche OSI (numéro)	***Couche OSI (nom)***	***Description***
7	application	Protocole SNIRIUM
6	présentation	Protocole SNIRIUM
5	session	Protocole SNIRIUM
4	transport	TCP
3	réseau	IP
2	liaison	Wifi + Ethernet
1	physique	Wifi + Ethernet

On choisit pour simplifier le traitement de la **requête** d'encoder l'ordre de pilotage sur un caractère, par exemple :

- **A** : Avancer
- **R** : Reculer
- **G** : tourner à Gauche
- **D** : tourner à Droite
- **S** : Stopper tout mouvement
- **?** : lire les valeurs des capteurs

Le robot pourra ensuite confirmer la trame reçue avec

```
OK     --> tout s'est bien déroulé
NOK    --> une erreur est survenue
```

Quand le robot doit transmettre toutes les valeurs des capteurs, il peut les regrouper dans une même trame et les séparer par un ;

```
angle;distance_obstacle;valeur_nsium;moteur_gauche;moteur_droite
```

Exemple de réponse avec des valeurs numériques :

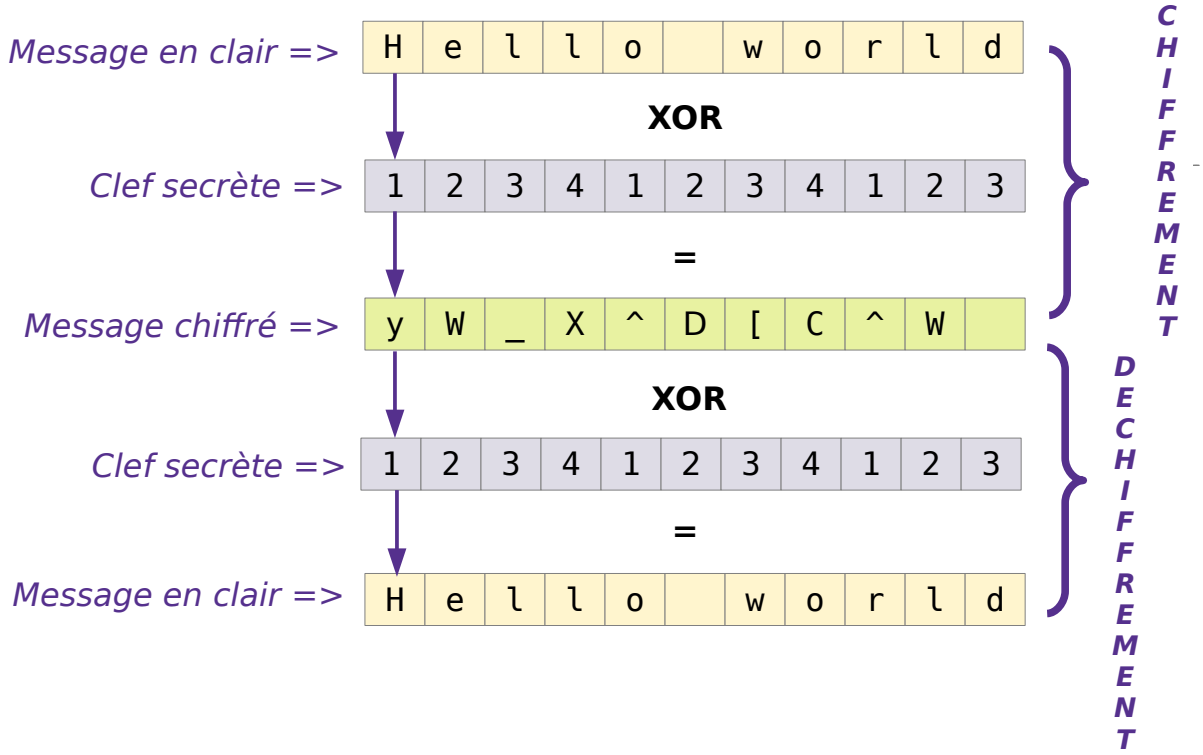
```
241;78;99;360;360
```

5.1.2 – Sécurité

Le chiffrement XOR est relativement aisé à mettre en œuvre en C++ (opérateur ^). Le principe est le suivant, pour chaque caractère :

```
H => 0x48 en ASCII => 0100 1000
                       XOR (^)
1 => 0x31 en ASCII => 0011 0001
                      ---------
                      0111 1001 => 0x79 en ASCII => y
```

1) code ASCII du message en clair :
2) code ASCII de la clef secrète : 1



L'opération de déchiffrement est la même que l'opération de chiffrement.

5.1.3 – Sockets

Voir cours/TP.

5.1.4 – Décodage de la requête

La requête pourrait arriver sur le serveur embarqué dans le robot comme ci-dessous.

```
A
```

Ainsi côté C++ cette requête qui arrive dans une variable de type string sera facile à exploiter en isolant le caractère à la position 20 dans le tableau de caractères :

```
string reponse(buffer);

if (reponse[0] == 'A') {
    cout << "Le robot avance " << endl;
}

if (reponse[0] == 'R') {
    cout << "Le robot recule " << endl;
}
```

5.1.5 – La réponse

Côté serveur il faudra répondre par une requête intégrant les valeurs issues des capteurs et encodées au format JSON comme ci-dessous :

```
241;78;99;360;360
```

Afin d'intégrer les valeurs issues des capteurs du robot (int ou float) dans la réponse (string), il faudra utiliser le formatage en mémoire avec **ostringstream** :

```
#include <iostream>
#include <sstream>

using namespace std;

int main()
{
    // Une valeur entière
    int une_valeur = 17;

    // Construction de la chaîne de caractères en mémoire
    ostringstream preparation;
    preparation << "La valeur est de " << une_valeur << " unité(s)";

    // Transformation de la chaîne préparée en string
    string chaine = preparation.str();

    // Affichage : La valeur est 17 unité(s)
    cout << chaine << endl;

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

5.2 – Pour l'IHM

5.2.1 – Les classes Qt

Le mini-projet GPS/Marathon a déjà été l'occasion de mettre en œuvre certaines classes du framework Qt que vous allez pouvoir réutiliser ici :

- ➔ La classe **QTcpSocket** permettra de communiquer en réseau avec le robot de manière asynchrone.
- ➔ La classe **QTimer** pourra elle permettre d'envoyer une requête pour récupérer les valeurs du robot à intervalle régulier.
- ➔ La classe **QPainter** vous permettra de dessiner sur une carte.

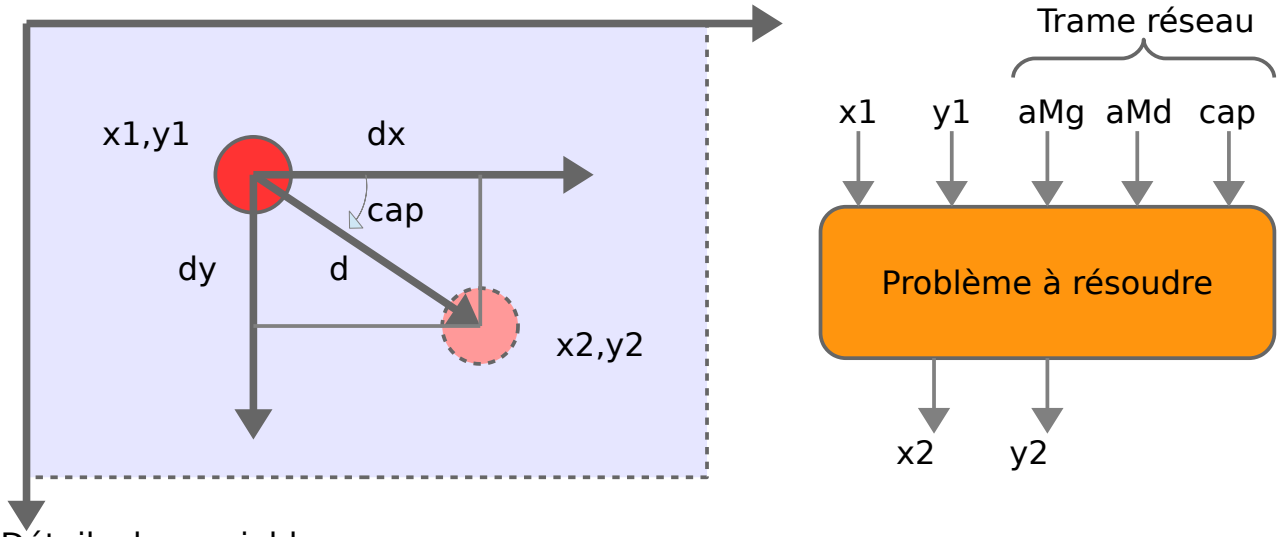
5.2.2 – Transformer les valeurs en carte

Une première approche simplificatrice consistera à considérer que le robot ne peut effectuer que certains mouvements :

- avancer ou reculer (les 2 moteurs avancent à la même vitesse)
- tourner sur place (les 2 moteurs tournent en sens inverse)

Ainsi on évite les mouvements en arc de cercle plus difficile à calculer et tracer.

Voici une modélisation dans le plan de la position du robot



Détails des variables :

- x1, y1 : position actuelle du robot
- x2, y2 : nouvelle position du robot
- aMg, aMg : respectivement angle du moteur gauche et droit
- cap : cap vers où « regarde » le robot

Algorithme de calcul de la nouvelle position du robot (idem pour obstacle)

```
// Ce qui est connu au début
diametre_roue = 56
x1 = 0
y1 = 0

// Ce qui est récupéré de la trame issue du robot
aMg = // angle du moteur gauche (issu du tachymètre)
aMd = // angle du moteur droite (issu du tachymètre)
cap = // cap du robot (issu du gyroscope)

// Ce qui est calculé
perimetre_roue = (2 * 3.14 * (diametre_roue / 2))
aMgMd = ((aMg - aMg_old) + (aMd - aMd_old)) / 2
d = (aMgMd * perimetre_roue) / 360
dx = d * cos(cap)
dy = d * sin(cap)
x2 = x1 + dx
y2 = y1 + dy

// Mémorisation des angles des 2 roues pour prochain calcul
aMg_old = aMg
aMd_old = aMd
```

La couleur du nouveau point est fonction du taux de snirium. Plus il est sombre, plus il y a de snirium dans le sol.

5.2.3 – La base de données

Ci-dessous une proposition de tables pour stocker les données :