Ky | Dehont | Stevan

Rapport : rover martien

1) Introduction

Avant d'explorer le sol de planètes extra-terrestres, il semble bon de se fixer des objectifs plus modestes pour apprendre les différentes compétences et connaissances nécessaires au bon déroulement d'une pareille mission. C'est précisément ce que nous propose ici la formation ingénieur de l'ISAE Supaero, au travers d'une démarche analytique et fonctionnelle visant à remplir les attentes d'un cahier des charges.

Ce document présente les différentes étapes de la conception de notre solution autant sur le plan logiciel que physique.

Dans un premier temps, nous nous pencherons sur l'analyse système et la conception fonctionnelle, au cœur de la démarche de ce module.

Ensuite, nous présenterons la solution dans son ensemble, aussi bien la partie logiciel que sa structure physique réalisée en *LEGO*.

Enfin, ce document regroupe divers tests que nous avons menés et résultats associés que nous avons récoltés.

2) Analyse système et conception fonctionnelle

Dans un premier temps, nous allons suivre la démarche à la base de la conception fonctionnelle. Ainsi, en partant du cahier des charges brut, nous pourrons avoir une meilleure idée de la manière de résoudre le problème qui nous a été posé.

a) De la finalité, de la mission et des objectifs

La première chose que nous devons faire est de nous intéresser aux racines du projet. Nous pourrons ainsi répondre aux questions "*quoi"*, "*pourquoi* "et "*comment"*.

- nous vivons dans un monde où des questions telles que "*quelles sont les origines de la vie sur Terre ?*", ou encore "*comment s'est formé notre système solaire ?*" sont au cœur des rêves et des défis des Hommes. **L'étude des corps constitutifs du système solaire, e.g. par retour d'échantillons, sur une planète comme Mars** apparaît être une finalité, autant sur les plans scientifique et technique que pour abattre la frontière jusqu'ici infranchissables qu'est l'espace.

- une approche classique et qui a fait ses preuves pour mener à bien de telles études est le rover. En toute généralité, un rover se doit de remplir plusieurs missions. Tout d'abord, notre rover doit être capable d'**explorer la zone d'intervention, parfois limitée en espace comme en temps**. Ensuite, il doit, dans cette zone, **localiser des échantillons de nature différente**. Certains petits, d'autres plus gros, de formes et de matières variées. Une fois localisés, le rover doit **se rendre sur place** et **récolter in situ ces échantillons**. Enfin, il doit être capable de **les ramener jusqu'à une zone de récupération définie** que ce soit pour une étude sur place ou un retour sur Terre.

- dans le but de mener à bien sa mission complexe, notre rover pourra remplir plusieurs objectifs plus simples. Il doit pouvoir ainsi **explorer la zone d'intervention avec des roues et des moteurs**. Puis, **localiser des échantillons avec des capteurs**. Chaque période de récolte d'échantillons doit être **terminée en moins de 7 minutes** et le rover doit pouvoir **en enchaîner 3 successivement**.

b) Le cycle de vie du rover

Avec les précédents points à l'esprit, nous pouvons désormais dresser le cycle de vie complet du rover dans son environnement simplifié pour l'exercice. Ce cycle est présenté dans la **figure 1** ci-dessous. Le cycle commence en haut à gauche avec le mode **atterrissage**, en pointillés gras. Le cycle est détaillé et s'arrête soit dans le mode **erreur** soit dans le mode **veille**.

Une séquence désigne le processus de récolte de deux échantillons. Une mission représente donc la réalisation de trois séquences.

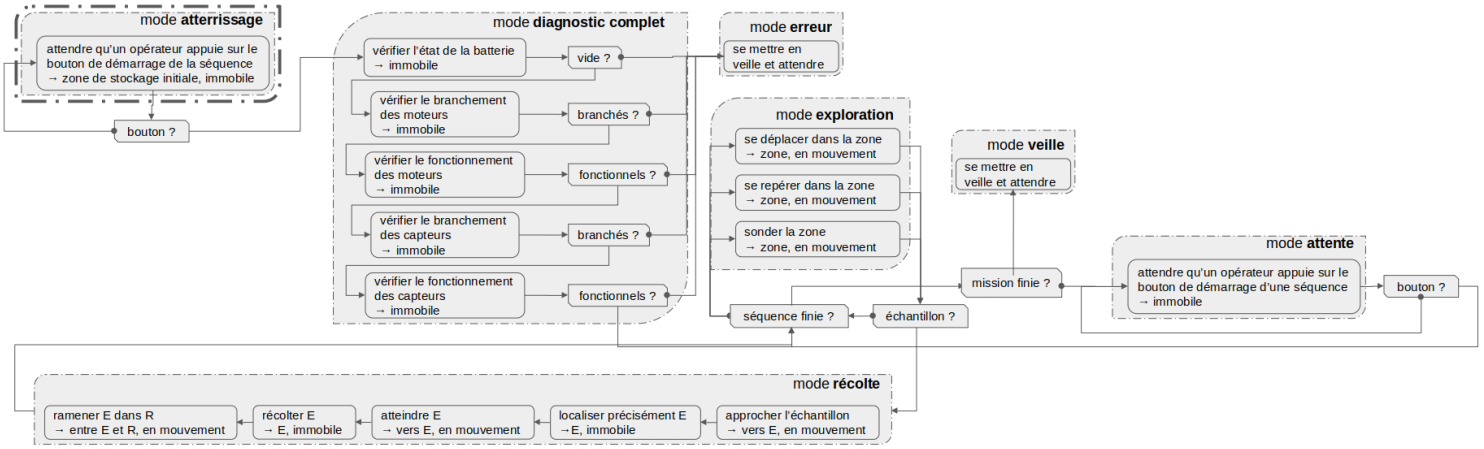

Figure 1 : cycle de vie du rover.

c) Les limites du système dans la phase opérationnelle

Une fois le **cycle de vie** idéal du rover dressé (voir section **2.c**), il nous faut nous intéresser aux différentes **limites** de notre futur système pendant sa **phase opérationnelle**.

Dans un premier temps, notre rover devra faire face à des contraintes physiques qui lui seront imposées par son environnement.

- la **zone d'intervention est ainsi limitée** dans l'espace ainsi que sa **géométrie** (voir section **3.a** pour plus de détails sur la zone). Le rover devra impérativement rester dans les limites géométriques de cette zone d'intervention, sous peine de se voir obligé de déclarer forfait pour la mission en cours. Il devra également faire abstraction des objets et obstacles en dehors de la zone. Il se peut en effet qu'une jambe ou un pied de chaise se trouve sous les feux des capteurs mais le rover ne devra en aucun cas essayer de récupérer de tels objets, en pensant que ce sont des échantillons.
- toutes les parties fournies dans la boîte de *Lego Mindstorms EV3* sont alimentées par une batterie rechargeable. Le rover devra ainsi s'assurer de la **bonne charge de ses batteries**. Cette contrainte n'est, dans notre cas précis, pas très forte. En effet, les batteries sont chargées avant chaque

séance et ne se déchargent pas en quelques heures d'utilisation, même intense. Cependant, on pourrait imaginer un rover qui indique un état critique de batterie et privilégie de rester proche d'une zone donnée plutôt que de partir en exploration avec des batteries faibles, au risque de se perdre et de ne pas pouvoir être rapatrié facilement.

- enfin, les capteurs et les moteurs doivent être branchés à la brique centrale par des câbles physiques. Le rover doit donc impérativement s'assurer du **bon branchement des capteurs et des moteurs** avant de commencer une mission. En effet, un seul capteur ou un seul moteur en moins rendrait n'importe quelle mission que le rover doit effectuer impossible, e.g. impossible de se repérer sans le capteur à ultrason, impossible de se déplacer avec un moteur en moins. On peut imaginer que de tels tests sont nécessaires si nous changeons des aspects du design physique du rover, un câble pourrait alors facilement se retrouver mal branché !

D'un autre côté, il y a des limites un peu plus diverses :

- le capteur de couleur a besoin d'être relativement proche des surfaces qu'il inspecte et ne disposant pas de suffisamment de moteurs pour le faire bouger, il y a un choix à faire entre :
    - détecter les marquages au sol → **blanc et noir uniquement**
    - détecter les couleurs des balles et de la zone de récupération→ rouge et blanc uniquement

  Faire la distinction entre les deux échantillons n'a pas vraiment d'importance, nous pouvons considérer que les seules couleurs à détecter sont le noir et le blanc.
- les missions proposées dans le cahier des charges imposent une contrainte temporelle : entre l'activation du rover et son immobilisation, il doit être capable de ramener les deux échantillons dans le zone de récupération **en moins de 7 minutes**.

d) Les fonctions et contraintes

Avec tous les éléments introduits plus haut, nous pouvons enfin dresser l'ensemble des fonctions et des contraintes que le rover devra prendre en compte. Dans cette partie, nous suivons la méthode de conception fonctionnelle enseignée cette année, i.e. le diagramme "en pieuvre". Le système est au centre et les éléments extérieurs sont tout autour. Une flèche simple représente une *fonction contrainte* alors qu'une flèche reliant deux éléments extérieurs en passant par le système représente une **fonction principale**.

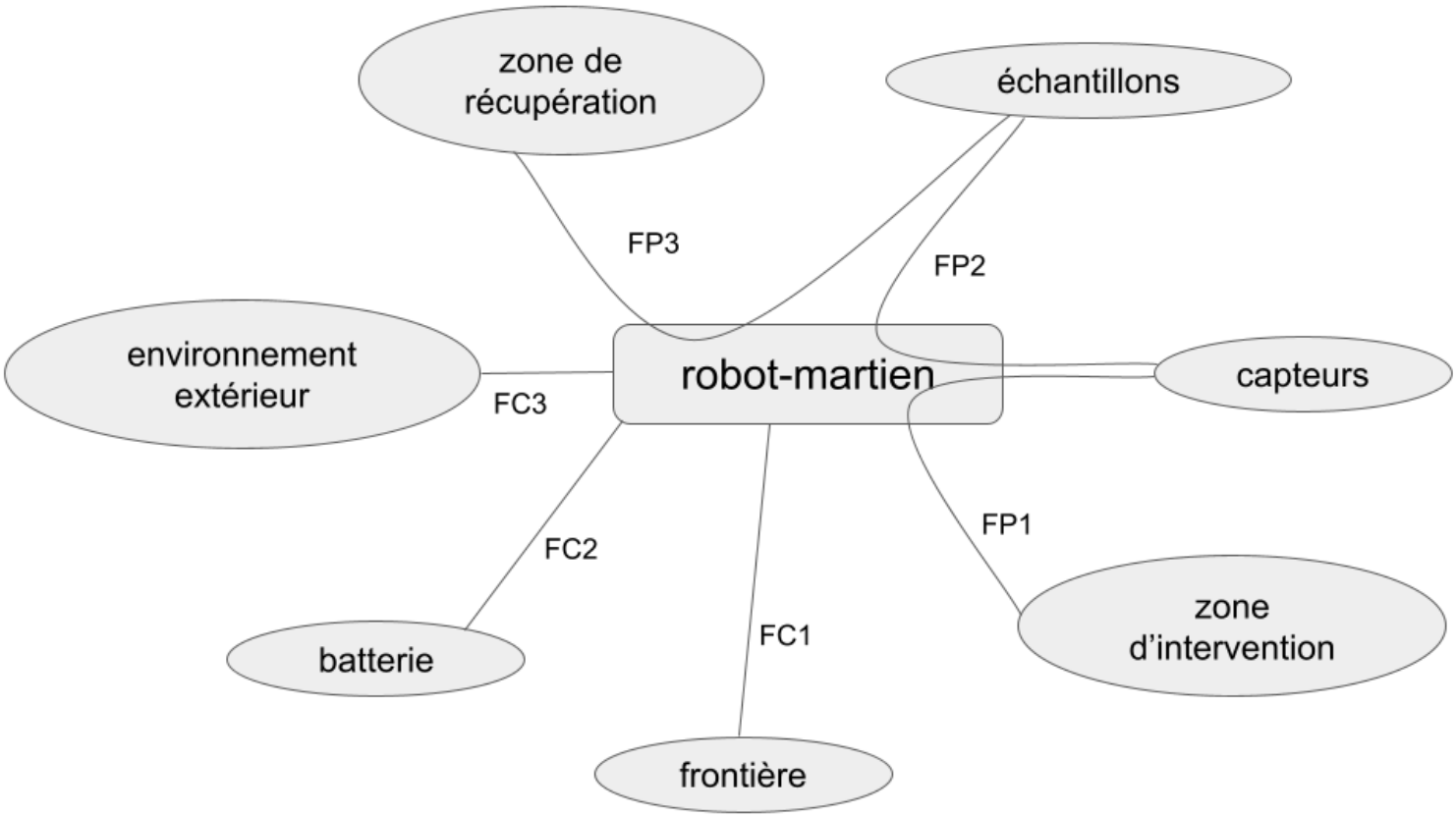

Figure 2 : diagramme "pieuvre" introduisant les différentes fonctions que le rover doit remplir. Les flèches simples représentent des *fonctions contraintes* alors qu'une flèche reliant deux éléments extérieurs en passant par le système représente une **fonction principale**.

FP1	se repérer dans la zone d'intervention.
FP2	localiser les échantillons.
FP3	transporter les échantillons vers la zone de récupération.
FC1	ne pas franchir la frontière de la zone d'intervention.
FC2	avoir suffisamment d'autonomie.
FC3	ne pas tenir compte des éléments extérieurs à la zone d'intervention.

Table 1 : liste des **fonctions principales** et des *fonctions contraintes* du rover.

e) L'architecture fonctionnelle

Nous avons à présent à disposition, le cycle de vie du rover (voir section **2.b**), les limites du rover imposées par l'environnement (voir section **2.c**) et les fonctions qu'il doit remplir pour accomplir sa mission (voir section **2.d**). Il nous faut maintenant décomposer les fonctions citées plus haut jusqu'à arriver à des sous-fonctions réalisables par un seul composant physique. Cette décomposition est détaillée dans les tableaux ci-dessous. Elle part de la liste des fonctions **principales** et *contraintes* de la section **2.d** et utilise le cycle de vie de la section **2.b** pour déduire les sous-fonctions.

Dans la suite, pour simplifier les notations, on notera E tout échantillon, D la zone de départ et R la zone de récupération.

Fonctions	Décomposition	Composants
FP1	**FT1.1** stocker une représentation de l'environnement	mémoires de l'ARM9
	FT1.2 mettre à jour cette représentation	
	FT1.2.1 observer l'environnement	
	FT1.2.1.1 détecter les frontières de la zone d'intervention	capteur de couleurs
	FT1.2.1.2 détecter les zones marquées au sol	capteur de couleurs
	FT1.2.1.3 détecter les reliefs	capteur à ultrason
	FT1.2.2 détecter une différence avec la représentation	processeur ARM9
	FT1.2.3 calculer une nouvelle représentation	processeur ARM9

Table 2 : décomposition fonctionnelle des fonctions principales suivantes :
FP1 - se repérer dans la zone d'intervention.

Fonctions	Décomposition	Composants
FP2	**FT2.1** explorer l'environnement	
	FT2.1.1 mettre le cap vers des zones non encore explorées	
	FT2.1.1.1 calculer les zones non encore explorées	processeur ARM9
	FT2.1.1.2 mettre le cap	mémoire de l'ARM9
	FT2.1.2 se déplacer vers une zone non encore explorée	
	FT2.1.2.1 transformer l'énergie électrique en énergie mécanique	moteur LEGO
	FT2.1.2.2 adhérer au terrain pour avancer	roues/pneus
	FT2.2 détecter un échantillon	capteur à ultrason
FP3	**FT3.1** récupérer E	
	FT3.1.1 s'assurer que E est bien en face du rover	capteur à ultrason
	FT3.1.2 saisir E	pince
	FT3.1.3 stocker E pour ne pas le perdre	pince
	FT3.2 se déplacer entre E et R	
	FT3.2.1 fixer le cap sur E	mémoires de l'ARM9
	FT3.2.2 rouler entre E et R, ou D et E	
	FT3.2.2.1 transformer l'énergie électrique en énergie mécanique	moteur lego
	FT3.2.2.2 adhérer au terrain pour avancer	roues/pneus
	FT3.3 déposer E dans R	
	FT3.3.1 s'assurer que le rover est bien devant R	capteur ultrason
	FT3.3.2 déposer E	pince

Table 3 : décomposition fonctionnelle des fonctions principales suivantes :

FP2 - localiser les échantillons.

FP3 - transporter les échantillons vers la zone de récupération.

Fonctions	Décomposition	Composants
FC1	*FT4.1* voir la frontière	capteur de couleurs
	FT4.2 commander l'arrêt des moteurs pour ne pas sortir	processeur ARM9
FC2	*FT5.1* stocker une quantité suffisante d'énergie	batteries
	FT5.2 contrôler le niveau de la batterie pour éviter les pannes	processeur ARM9
FC3	*FT6.1* détecter des objets extérieurs	capteur à ultrason
	FT6.2 déterminer que ces objets n'ont pas d'intérêt pour la mission	processeur ARM9
	FT6.3 ne plus considérer cette zone de l'espace	mémoires de l'ARM9

Table 4 : décomposition fonctionnelle des fonctions contraintes
FC1 - ne pas franchir la frontière de la zone d'intervention.
FC2 - avoir suffisamment d'autonomie.
FC3 - ne pas tenir compte des éléments extérieurs à la zone d'intervention.

f) L'architecture physique

Une fois la liste des composants nécessaires à la réalisation des fonctions introduites précédemment, il nous est possible de représenter une architecture abstraite du rover final.

On remarque immédiatement une chose : tous les éléments sont connectés entre eux par le biais de la brique *Lego Mindstorms EV3*.

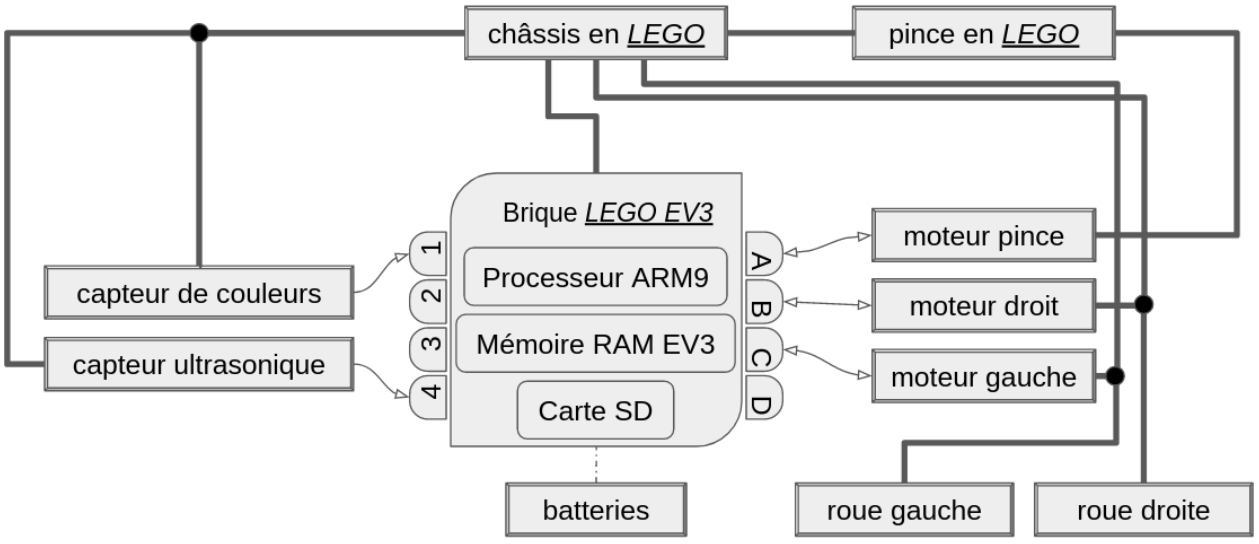

Figure 3 : architecture physique abstraite du rover. Tous les composants nécessaires à la réalisation des fonctions sont présents, les mémoires de l'ARM9 ont été séparées en deux sous mémoires, pour rendre mieux compte de la réalité, même si on peut considérer indifféremment la RAM et la carte SD.

- une flèche fine représente un câble de connexion. Le sens de la flèche indique le sens de transfert des informations, e.g. on peut lire un capteur, on peut écrire sur un moteur (lui donner un ordre de mouvement) ou bien lire (récupérer la position du moteur en degrés).

- une ligne épaisse représente une connexion physique entre deux composants, réalisée par des pièces *LEGO*, e.g. la brique est fixée au châssis, le moteur de la pince est fixé à la pince …
- une ligne en pointillée représente enfin une connexion énergétique, i.e. entre les batteries et la brique. Les échanges d'énergie entre la brique et les autres éléments connectés ne sont pas représentés pour ne pas surcharger le schéma.

g) La matrice de conformité

Enfin, et ce sera le dernier élément de l'analyse fonctionnelle du système "rover", nous sommes en mesure de dresser la matrice de conformité de notre système. C'est un résumé de toutes les parties précédentes et qui fait le lien entre chaque sous-fonction et le composant qui la réalise.

Fonction	Microprocesseur ARM9		Mécanique		Capteurs		batteries	pince
	mémoires	calculateur	moteur	roues	couleurs	ultrason		
FT1.1	x							
FT1.2.1.1					x			
FT1.2.1.2					x			
FT1.2.1.3						x		
FT1.2.2		x						
FT1.2.3		x						
FT2.1.1.1		x						
FT2.1.1.2	x							
FT2.1.2.1			x					
FT2.1.2.2				x				
FT2.2						x		
FT3.1.1						x		
FT3.1.2								x
FT3.1.3								x
FT3.2.1	x							
FT3.2.2.1			x					
FT3.2.2.2				x				
FT3.3.1						x		
FT3.3.2								x

Table 5 : matrice de conformité des **fonctions principales**.

Fonction	Microprocesseur ARM9		Mécanique		Capteurs		batteries	pince
	mémoires	calculateur	moteur	roues	couleurs	ultrason		
FT4.1					x			
FT4.2		x						
FT5.1							x	
FT5.2		x						
FT6.1						x		
FT6.2		x						
FT6.3	x							

Table 6 : matrice de conformité des *fonctions contraintes.*

3) Présentation du robot

Passons maintenant à la solution concrète à nos problèmes de récolte d'échantillons. Nous verrons tout d'abord plus précisément les exigences et les contraintes auxquelles le rover doit faire face. Ensuite nous parlerons des parties logiciel et structure du rover. Enfin nous présenterons les différents plans de validation ainsi que les résultats obtenus.

a) Les exigences et les contraintes

En repartant du cahier des charges et de ce qui a été écrit plus tôt dans ce document, nous pouvons faire la liste des différentes contraintes, qu'elles soient physiques, géométriques, logicielles ou temporelles :

- le rover part de la **zone de stockage initiale**, on peut supposer que la position initiale est parfaitement connue et reproductible ou bien la recalculer au début des missions au cours d'une étape de calibrage.
- le rover **attend** dans cette zone **l'autorisation d'un opérateur** extérieur. Il doit donc être prêt à partir dès que le bouton est enfoncé.
- le rover doit alors **remplir sa mission**, déjà décrite plus haut, i.e. localiser et ramener deux échantillons en moins de 7 minutes dans la zone de récupération.
- le rover doit **s'arrêter automatiquement** près de la zone de récupération une fois les deux échantillons ramenés.
- le rover doit **enchaîner trois missions successives**, pour un total de 21 minutes maximum.

Il existe également des contraintes et exigences sur la zone elle-même et l'environnement autour du rover pendant les missions :

- la zone d'intervention est **(quasi) plane, horizontale, rigide, de couleur claire et quasi uniforme**. Les dimensions sont connues et égales à : **2500 mm** pour le grand côté et **1500 mm** pour le plus court, pour une surface totale de 3.75 m².
- les conditions de températures, de pression et d'hygrométrie sont également connues:
    - température entre 15°C et 32°C
    - pression entre 1000 et 1030 mb
    - hygrométrie entre 40% et 75%

on remarquera que les composants à disposition fonctionnent de manière nominale dans ces conditions atmosphériques, on n'aura pas à s'en soucier dans la suite, comme notre rover ne quittera jamais ces conditions.

- la zone de stockage initiale est en bas au centre de la zone d'intervention et est un **carré de côté 500 mm, de couleur noire**
- la zone est délimitée par une **bande de largeur 50 mm** qui fait tout le tour, i.e. sur chaque côté du rectangle définissant la zone.
- la zone de récupération est, quant à elle, un **disque rouge de 200 mm de diamètre**, délimitée par un **cercle rouge de 40 mm de hauteur, et dont l'épaisseur se situe entre 1 mm et 2 mm**.

Enfin, les échantillons eux-mêmes apportent leurs lots d'exigences et de contraintes :

- échantillon 1 : sphère de **50 mm de diamètre**, de couleur **rouge**, sur un **support noir**
- échantillon 2 : sphère de **39 mm de diamètre**, de couleur **blanche**, sur un **support gris** (les couleurs des supports sont en fait très proches l'une de l'autre et plutôt gris sombre)

La géométrie de la zone est synthétisée dans la figure suivante.

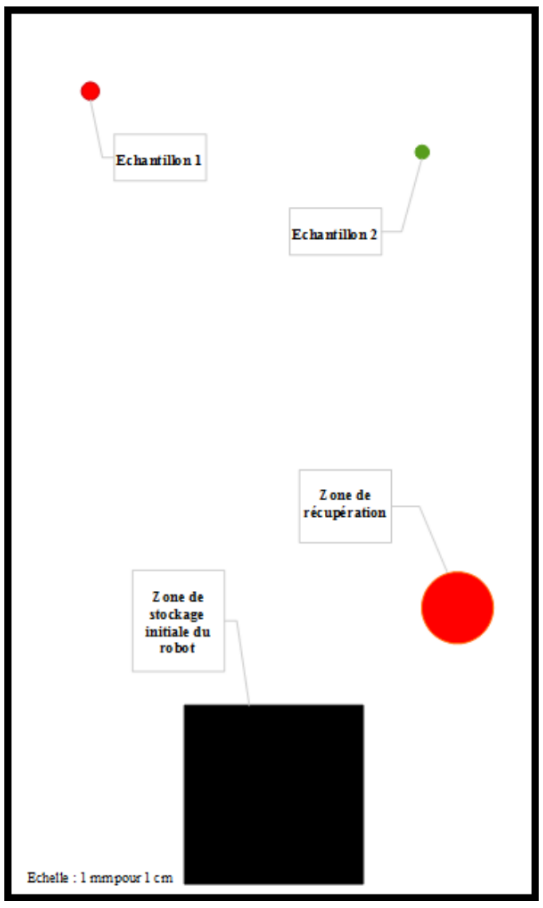

Figure 4 : schéma de la zone d'intervention, avec les deux échantillons placés.

b) Le logiciel

Pour expliquer les aspects logiciels de notre solution, le plus simple semble être de décrire un cycle de vie complet du rover. Prenons donc les étapes les unes après les autres :

1. connexion des périphériques → constructions des objets en mémoire
2. vérification de la batterie → essentiellement des *if* et des *print*
3. algorithme principal d'enchaînement des missions.
   - 3.1. for i = 1 to 3 do;
   - 3.2. *initialize obstacle detection*[4]
   - 3.3. while not (*mission is done*[5]) do;
   - 3.4. *explore*[6]
   - *3.5.* *harvest*[7]
   - 3.6. done
   - 3.7. *check batteries level*
   - 3.8. *wait*[8]
   - 3.9. done
4. *initialize obstacle detection* → remets à zéro un tableau interne d'obstacles ainsi que l'indice de l'obstacle en cours.
5. *mission is done* → vérifie si l'indice de l'obstacle courant est égal à 2
6. *explore* → voir section **3.b.i**
7. *harvest* → voir section **3.b.ii**
8. *await* → arrête l'exécution du programme et attend qu'un opérateur appuie sur un bouton pour relancer une nouvelle mission.

i) La méthode *explore*

Dans le cahier des charges, il est précisé que les échantillons sont placés aléatoirement dans la zone d'intervention. Il nous a donc fallu trouver une stratégie d'exploration permettant au rover de ne pas passer à côté d'un seul échantillon. Pour ce faire, nous avons opté pour une exploration exhaustive de la zone. Ce choix est motivé dans les sections **3.d** et **3.e** où nous nous sommes rendu compte qu'il est possible de faire se déplacer le robot tout en scannant les alentours avec les capteurs. Cette découverte nous a donc poussé à chercher à explorer toute la zone, le rover pouvant alors tout à fait ne pas perdre trop de temps en balayant le grand rectangle de la zone d'intervention.

Dans un premier temps, nous avons eu l'idée de faire suivre le rover un chemin fermé balayant l'ensemble de la zone (voir figure 6). Les différents points de passage détaillés ci-dessous sont calculés dans la méthode *compute_path* de la classe *Rover*.

La stratégie d'exploration est alors la suivante :

```
tant que (waypoint_index < path.length-1) faire
    rotation vers le point de passage path[waypoint_index+1] non bloquante
    tant que (rotation en cours) faire
        distance <-- mesure ultrasonique
        // calcule des coordonnées de l'objet avec la position du rover et la distance à l'objet
        objet <-- object_from_distance
        si (objet dans la zone) et (objet pas encore vu) et (objet pas dans la zone de récup) alors
            sortir de explore pour appeler harvest
        fin si
    fin tant que

    translation vers le point de passage path[waypoint_index+1] non bloquante
    tant que (translation en cours) faire
        distance <-- mesure ultrasonique
        objet <-- object_from_distance
        si (objet dans la zone) et (objet pas encore vu) et (objet pas dans la zone de récup) alors
            sortir de explore pour appeler harvest
        fin si
    fin tant que
    waypoint_index <-- waypoint_index + 1
fin tant que
```

Figure 5 : pseudo-code de la méthode *explore* de la classe *Rover*.

On remarque que, si un objet est détecté pendant une rotation -pour s'aligner avec le prochain point de passage- ou pendant une translation -pour rejoindre la prochain point de passage- alors le rover quitte la méthode *explore* et exécute la méthode *harvest* décrite dans la section **3.b.ii**. En regardant l'algorithme principal décrit dans la section **3.b**, on voit que les méthodes *explore* et *harvest* sont exécutées en alternance, si bien qu'au prochain appel de *explore*, le parcours du chemin reprendra au dernier point de passage, jusqu'à trouver un nouvel échantillon, ou bien terminer le chemin en entier, auquel cas la mission est terminée.

Figure 6 : chemins permettant de balayer l'ensemble de la zone. Ici la valeur de x est choisie très grande pour une meilleure lisibilité du schéma, mais elle est réglable dans le code. La valeur à choisir est donc celle qui permet le meilleur balayage, sans perte de temps à scanner des zones déjà connues. Le chemin commence en bas à droite pour toutes valeurs de x et balaye la zone d'intervention dans la longueur. La fin du chemin dépend de la valeur de x.

ii) La méthode *harvest*

Entre deux morceaux de la phase d'exploration, le rover est amené à récolter des échantillons, par exemple lorsqu'un nouvel obstacle dans la zone exploitable est détecté pendant *explore*. La stratégie est ici relativement simple.

D'après le pseudo-code de la méthode *explore*, on sort de celle-ci dès qu'une mesure concluante est faite. Utilisons cette information pour débuter *harvest* :

```
on pose approach = true
on pose factor = 0.5
tant que (approach) faire
    distance = mesure ultrasonique
    si (distance = infini) alors
        perdu = true
        pour angle allant de -check à +check faire
            rotation jusqu'à l'angle rover.heading+angle
            distance = mesure ultrasonique
            si (distance < infini) et (echantillon exploitable) alors
                perdu = false
            fin si
        si (perdu) alors
            approach = false
        fin si
    sinon
        si (echantillon exploitable) alors
            si (distance < distance_securite) alors
                approach = false // on est suffisamment proches
            sinon
                avancer de factor*distance en direction de l'échantillon
            fin si
        fin si
    fin si
fin tant que

// on est en face de l'échantillon
noter la position de l'échantillon
// pour ne plus le récolter si le rover le rencontre à nouveau
ouvrir la pince
avancer de (distance - distance entre le centre de rotation et la pince)
// pour aligner la pince et l'échantillon
fermer la pince
rotation pour s'aligner avec la zone de récupération
translation pour approcher la zone de récupération
ouvrir la pince // pour lâcher l'échantillon
reculer
fermer la pince
```

Figure 7 : pseudo-code de la méthode *harvest* de la classe *Rover*.

c) La structure du robot

La conception de la structure de notre rover s'est faite en plusieurs étapes. Dans un premier temps, nous avons trouvé deux éléments, un modèle de châssis et un modèle de pince (voir le document **./rapport/track3r-manual.pdf**, pages **3-27 pour le châssis** et pages **53-67 pour la pince**). Cependant malgré l'esthétique du rover ainsi construit, nous avons rencontré un certain nombre de difficultés qui nous ont longtemps ralenti pendant le projet.

- tout d'abord, le modèle exact présent dans le document **./rapport/track3r-manual.pdf** n'était pas adapté à la récolte d'échantillons. En effet, avec la conception du châssis initiale, la pince semble avoir été conçue pour attraper des choses à même le sol. Or, nous voulions attraper des balles à quelques centimètres de hauteur. Il a donc fallu rehausser la pince et en particulier la brique *LEGO*. Pour ce faire, nous avons simplement déporté les moteurs directement sous la brique. Pendant ce processus, il a été nécessaire de reconstruire l'intégralité du châssis. Finalement, nous n'avons pas utilisé le manuel **./rapport/track3r-manual.pdf** pour le design final, mais nous nous en sommes inspiré, notamment pour l'utilisation des chenilles et de la pince.
- ensuite, la pince était légèrement trop basse pour passer au-dessus de la limite circulaire de la zone de récupération. En effet, le moteur contrôlant la pince semblait tomber légèrement en avant sous son propre poids. Nous avons donc renforcé les différentes liaisons sur cette zone, fixant ainsi les pinces et le moteur dans leur position finale (voir **./rapport/structure-rover.pdf** pour plus de détails, notamment sur les photos vues de dessus).
- le dernier problème majeur que nous avons rencontré au niveau de la structure est celui des chenilles. Cette idée nous était apparue plutôt bonne sur le papier. Toutefois, il aurait été préférable de disposer de chenilles en caoutchouc car les glissements trop fréquents du robot ont nécessité beaucoup de corrections dans les calculs de trajectoire, ce qui a terriblement affecté la localisation du rover dans l'espace et l'efficacité des tests et de la conception logiciel. Afin de remédier à ce problème, nous nous sommes orientés vers une architecture plus simple avec deux roues en caoutchouc. Cela nous a permis de minimiser les dérapages en améliorant l'adhérence et ainsi de pouvoir bénéficier d'un meilleur repérage sur la map.

En conclusion, ce sont ces divers problèmes qui ont permis de forger la structure finale de notre rover. La dernière version du design est détaillée dans le document **./rapport/structure-rover.pdf**. Pour permettre une lecture plus simple de ce pdf, voici quelques indications pour le lecteur :

- l'ordre des photographies est le suivant :
    - vue de côté
    - vue de derrière
    - vue de devant, pince fermée
    - vue de devant, pince ouverte droite
    - vue de devant, pince ouverte gauche
    - vue de dessus
    - vue de dessous, support de pince
    - vue de dessous, châssis
- l'axe des x est en rouge
- l'axe des y est en vert
- l'axe des z est en bleu
- les axes ne sont pas normés pour faciliter la lecture
- pour plus de précision, les axes devraient être fixés dans le coin en bas à gauche de la zone mais sont ici placés au mieux sur le rover pour faciliter le repérage dans l'espace.
- le centre de rotation du rover est situé entre les deux roues. Les pages 1, 2 et 3 placent d'ailleurs le système d'axes au niveau de ce centre de rotation.

d) Le plan de validation

Pour valider notre solution, nous avons effectué de nombreux tests au fur et à mesure du déroulement du projet :

- le **capteur ultrason** → utilisation de la méthode *test_ultrasonic_sensor* de la classe *Rover* pour calculer la taille angulaire du cône de détection ainsi que la portée du capteur. Cette méthode effectue des mesures de distance ultrason en permanence et affiche le résultat sur l'écran de la brique *LEGO* tant que l'on n'appuie pas sur le bouton central.
- le **capteur de couleur** → utilisation de la méthode *test_color_sensor* de la classe *Rover* pour vérifier les codes couleur fournis par la documentation ainsi que la sensibilté au noir et blanc, les deux couleurs que nous étions amenés à détecter. Cette méthode effectue des mesures de couleur au sol en permanence et affiche le résultat sur l'écran de la brique *LEGO* tant que l'on appuie pas sur le bouton central.
- les **moteurs** → avec les méthodes *test_motors* et *test_navigator** de la classe *Rover*, nous voulions tester les capacités des classes leJOS à remplir les tâches dont nous avions besoin pour les missions. Ces méthodes demandent aux moteurs d'effectuer un certain nombre de rotations, relatives ou absolues, pour vérifier leur bon fonctionnement.
- la **géométrie des roues du rover** → pour permettre des mouvements cohérents avec les ordres donnés dans le code du rover, il faut donner des mesures assez précises de l'entraxe des roues ainsi que de leur diamètre. Il suffit de faire se déplacer le rover d'une distance connue, en ligne droite, pour ajuster le diamètre des roues, e.g. si la distance réelle parcourue est trop grande, le diamètre des roues donné au code est trop petit, il faut donc l'augmenter. Une fois le diamètre des roues réglé, il faut s'intéresser à l'entraxe pour les rotations. On procède de la même manière que précédemment mais avec des rotations et des réglages sur l'entraxe, e.g. si l'angle réel de rotation est trop grand, l'entraxe donné au code est trop grand, il faut le diminuer.
- la **pince** → nous avons utilisé les méthodes *release* et *grab* de la classe *Grabber* afin de calibrer l'angle d'ouverture nécessaire à la pince pour saisir les échantillons et la vitesse du moteur lié à la pince pour réaliser cette action sans faire tomber les échantillons. Ces tests ont également été réalisés afin de s'assurer que la hauteur de la pince permettrait la saisie des échantillons tout en étant assez haute pour passer le cercle de délimitation de la zone de récupération.
- des **tests unitaires en début de mission** → cette partie est détaillée dans la section **3.b** et consiste simplement en des constructeurs de classe Java qui renvoient des erreurs lorsque les capteurs ou moteurs sont mal branchés.
- la **stratégie d'exploration** : le principe est de lancer la méthode *explore* de la classe *Rover* sans se soucier d'appeler la méthode *harvest*. On simule ainsi le balayage de la zone tout en ne se concentrant que sur celui-ci.
- la **stratégie de récolte** : de même que pour la phase d'exploration, le principe est ici de lancer la méthode *harvest* de la classe *Rover* en donnant au rover un échantillon dont la position relative est connue. On simule ainsi la récolte d'échantillons en ne se concentrant que sur celle-ci.

Enfin, des tests plus complets pour vérifier le bon fonctionnement de tout le logiciel :

- ➢ **lancement d'une sous-mission**, i.e. récupération de 2 échantillons en partant d'un état d'attente dans la zone de stockage initiale, puis immobilisation finale quand les deux échantillons sont récoltés.
- ➢ **lancement d'une mission complète**, i.e. récupération de 2 échantillons, 3 fois de suite pour vérifier que l'enchaînement des différentes sous-missions ne pose pas de problème d'exécution.

e) résultats de tests

La liste de tests présentée ci-dessus dans la section **3.d** a été dressée au fur et à mesure. Voici, dans le même que précédemment, des résultats de ces différents tests :

- ➢ le **capteur ultrason** → en fixant la position du rover, en particulier celle du capteur ultrason, nous avons effectué des mesures en positionnant des objets à différentes distances et angles, relativement à l'axe du capteur. Nous avons ainsi obtenu quelques résultats :
  - ○ le capteur semble être capable de prendre des mesures pour des distances allant de 15 cm environ à près de 1.5 m. Pour plus de sécurité, comme les mesures perdent en précisions dans ces plages de distances extrêmes, nous avons choisi de prendre un plage de valeur allant de **20 cm à 1 m**.
  - ○ le capteur est capable de "voir" des objets se trouvant légèrement sur le côté. Les mesures sont tout à fait possible en face du capteur (angle nul) et sont possibles également pour un angle allant jusqu'à environ 15° de part et d'autre du capteur. Comme les mesures perdent en précision pour ces valeurs d'angles extrêmes, nous avons choisi une plage angulaire avec plus de marge, i.e. des angles allant de **-10° à +10°** pour un cône de détection avec une **ouverture angulaire de 20°**.
- ➢ le **capteur de couleur** → nous avons rapidement vérifié les valeurs des codes couleurs pour le blanc et le noir, qui coïncidaient avec les valeurs données dans la documentation. Cependant, nous nous sommes rendu compte que, étant données les deux seules couleurs à détecter, il était préférable de prendre des mesures de luminosité. En effet, que ce soit avec ou sans la lumière rouge activée sur le capteur de couleur, le noir renvoyait toujours des valeurs en dessous de 0.5 alors que le blanc renvoyait toujours des valeurs au dessus de 0.5. Pour être plus précis, le noir était plutôt en dessous de 0.2 et le blanc au dessus de 0.6. Et, avec la lumière rouge désactivée, le noir descend en dessous de 0.1 et le blanc monte au dessus de 0.7. Nous avons donc considéré une valeur seuil assurant une bonne distinction entre le noir et le blanc, i.e. un **seuil de 0.5 sur la luminosité**.
- ➢ les **moteurs** → rapidement, à l'instar de plusieurs autres groupes travaillant sur le même projet, nous nous sommes rendus compte que les classes leJOS assez haut niveau commandant les moteurs étaient assez peu adaptées à notre usage. Le comportement nous a paru assez imprévisible, notamment lorsqu'il faut utiliser les calculs de *Pose*, e.g. avec la méthode *getPose* de l'interface *PoseProvider*. Nous avons pensé qu'il serait plus rapide de coder quelques méthodes de déplacement qui ne se basent pas sur la méthode *getPose* de l'interface *PoseProvider.* Cette phase a été assez rapide et nous avons pu avancer après coup.
- ➢ la **géométrie des roues du rover** → comme présenté dans le test (voir section **3.d**), nous avons ajusté les valeurs de diamètre des roues et d'entraxe pour avoir des mouvements collant le plus possible aux valeurs dans le code. Nous avons alors des **roues de 55 mm de diamètre** et un **entraxe de 107 mm**.

- ➢ la **pince** → nous avons pu déterminer par ces tests que la pince avait besoin d'une **ouverture angulaire de 40°** et d'une **vitesse de saisie de 10°/s.**
- ➢ des **tests unitaires en début de mission** → cette partie fonctionne plutôt bien. Cependant, quelques points sont restés non résolus pendant le projet :
    - sans toucher aucun des capteurs -ces derniers ont des prises type RJ45 avec une sécurité pour éviter les déconnexions- les constructeurs des classes associées renvoient parfois des erreurs, ce qui bloque le programme dans le mode *erreur*, i.e. LEDs clignotantes rouges. Il suffit alors en général de relancer le code pour corriger le problème.
    - quant à eux, les tests sur les moteurs ne marchent pas du tout. Qu'ils soient branchés ou non, les constructeurs associés ne renvoient pas d'erreur. Ce qui peut se comprendre car les moteurs sont surtout des périphériques sur lesquels on accède en écriture, contrairement aux capteurs qui sont en lecture et qui ont donc besoin de répondre dès le constructeur de classe. Cependant, même en ajoutant des appels de méthodes en écriture sur les moteurs, ces dernières ne renvoient rien, quel que soit le branchement physique. Enfin, même en utilisant une des seules méthodes en lecture, i.e. *getTachoCount* de l'interface *RegulatedMotor*, nous ne sommes pas parvenus à obtenir d'erreur sur des branchements invalides. Nous avons donc décidé de passer à la suite en faisant attention à bien brancher les moteurs pour ne pas avoir de mauvaises surprises.
- ➢ la **stratégie d'exploration** : les précédents tests ont été tous assez concluants. Cependant, ils nous ont pris beaucoup de temps, ce qui nous a laissé un temps assez limité pour les derniers gros tests importants. Pour la stratégie d'exploration, le calcul du chemin semblait fonctionner car le rover suivait une trajet similaire à celle présentée sur la figure 6 dans le section **3.b.i**. Cependant, nous avons découvert un gros problème qui a plus ou moins marqué la fin du test sur le stratégie d'exploration : le rover n'avance pas du tout en ligne droite. Il est toutefois plutôt symétrique, et les violations de symétries viennent de pièces en lego dont la masse est très négligeable devant celles de la brique, des moteurs et des capteurs, qui eux respectent tous la symétrie gauche/droite. Nous n'avons donc pas eu le temps de trouver une solution pour remédier à ce décalage, qui est d'ailleurs important, i.e. en quelques centimètres, le décalage est déjà légèrement visible, et il s'amplifie sur les 2.5 m de longueur de la zone, ce qui rend le chemin obsolète et perd le rover dans la zone.
- ➢ la **stratégie de récolte** : nous avons eu un peu de temps pour tester notre stratégie de récolte. Cette dernière, après quelques réglages et modifications sur la préhension de la pince notamment, a été plutôt concluante sur notre table de travail, e.g. le rover a attrapé la balle plusieurs fois. Cependant, une fois sur la zone d'intervention, les performances n'étaient plus au rendez-vous. Nous pensons que le rover captait des éléments extérieurs avec les ultrasons, mais nous n'avons malheureusement pas eu le temps de trouver de corriger ce problème sur la zone d'intervention.

Enfin, pour les tests plus complets, i.e.le **lancement d'une sous-mission** et le **lancement d'une mission complète**, nous n'avons simplement pas eu le temps de déployer le rover. Le code principal qui contrôle les missions étant assez simple, il y a des chances qu'il ait pu marcher rapidement, mais avec les étapes précédentes, i.e. la **stratégie d'exploration**

et la **stratégie de récolte**, trop peu concluantes, les missions plus ou moins complètes n'ont pas du tout fonctionné.

4) Conclusion

Nous pouvons revenir dans cette dernière partie sur les différents points que nous auront appris ce projet, le travail en équipe et le sujet précis de notre travail.

Étant tous passés par la compétition robotique en première année, la grande nouveauté pour nous était la découverte de la bibliothèque leJOS, incluant des classes conçues spécialement pour l'EV3. En effet, la principale différence avec la compétition robotique de la première année à Supaero est sur le code lui-même, la structure physique en *LEGO* étant exactement identique. Ici, nous devions coder en Java avec les programmes de la bibliothèque leJOS, contrairement à la programmation en bloc de la compétition robotique. Ce choix nous a permis de grandement gagner en liberté, nous avons en effet pu écrire des morceaux de code plus adaptés à notre usage, choisir la conception orientée objet de notre choix et gagner en compétences dans l'utilisation de la brique EV3.

Nous avons appris, quelque peu à nos dépens, que le travail en équipe est un art délicat, dans lequel il est facile de se perdre un peu. C'est probablement ce qui explique pourquoi nous n'avons pas eu le temps de faire des tests sur les dernières méthodes que nous avons écrites. Nous avons ainsi perdu du temps sur la répartition du travail et sur les tests. Nous aurions pu, ou dû, paralléliser plus le travail au niveau du code pour permettre d'avancer plus efficacement, cependant nous avons pris conscience du retard que nous avions pris un peu tard. En somme, les méthodes ont été codées à temps mais les tests n'ont pas pu rentrer dans l'agenda serré du projet et de la compétition finale.

Malgré ces manquements, nous avons appris de nombreuses choses. Nous avons appris à utiliser la bibliothèque leJOS, comme dit précédemment, mais nous avons aussi tiré des enseignements sur le travail d'équipe et la gestion de projet, comme suggéré plus haut, et nous avons également appris des choses sur la conception de "vrais" rover spatiaux, notamment avec la présentation du CNES nous montrant énormément d'éléments de conception, autant algorithmiques que physiques, auxquels nous n'avions pas du tout pensé !