



**POLYTECHNIQUE  
MONTRÉAL**

**UNIVERSITÉ  
D'INGÉNIERIE**

Département de génie informatique et génie logiciel

INF3995

**Projet de conception d'un système informatique**

## Protocole de tests

### ***Conception d'un système d'exploration spatiale***

Équipe No 102

Majeed Abdul Baki

Étienne Perron

Julien Roux

Victor Gilbert

Abdel Madjid Sant'anna

Étienne Hourdebaigt

Février 2024

## Table des matières

1.	Protocole de tests .....	3
2.	Tests par requis.....	3
a.	Requis identifier (R.F.1) .....	3
b.	Requis commandes de missions (R.F.2) .....	4
c.	Requis états des robots affichés en temps réel dans l'interface (R.F.3) .....	4
d.	Requis exploration et évitement d'obstacles (R.F.4 et R.F.5) .....	4
e.	Requis clients en simultanés et support (R.F.10) .....	5
f.	Affichage de la carte généré par les robots (en cours) (R.F.8) .....	5
g.	Requis modification du code et mise à jour des robots (R.F.14 et R.F.16) .....	6
h.	Enregistrement des données de mission dans la BD (R.F.17- R.F.18) .....	6
i.	Requis affichage des logs et informations envoyées par les robots (R.C.1).....	7
j.	Requis générations de murs aléatoires (R.C.3) .....	7

## 1. Protocole de tests

La nature embarquée du projet rend difficiles les tests des composantes embarqués de notre système, il est néanmoins nécessaire de mettre en place des protocoles clairs pour vérifier des potentiels dysfonctionnements et régressions des fonctions du robot.

Ainsi pour chaque nouvelle fonctionnalité développée, il faut dans un premier temps vérifier que tout compile bien, puis dans la simulation on doit tester notre nouvelle fonctionnalité pour vérifier son comportement et qu'il correspond bien à celui attendu.

Une fois que le comportement en simulation est satisfaisant, on vérifie que les fonctionnalités précédentes fonctionnent toujours correctement (voir partie : Tests par requis).

Enfin, une fois que tout est valide sur simulation, on peut l'essayer sur le robot physique.

Néanmoins la réalité du développement oblige parfois d'accélérer certaines étapes, il est donc possible de passer les tests de régressions et passer directement des tests physiques. Dans ce cas, la validation des requis précédents sera faite avant la *merge request* et l'intégration du nouveau code dans la branche principale.

## 2. Tests par requis

Dans cette partie nous allons décrire les étapes pour valider le fonctionnement de chaque requis, étant donné la présence de tests unitaires pour le client et le serveur, ces tests visent principalement à vérifier le comportement du robot, mais ils permettent aussi de réaliser des tests de bout en bout des fonctionnalités ce qui consolide leur validation.

En raison des performances du réseau, une latence de quelques secondes entre l'activation et le retour d'une fonction est acceptable pour le robot physique.

### a. Requis identifier (R.F.1)

On veut vérifier que lorsque l'on appuie sur l'un des boutons **identifier** de notre client, uniquement le robot correspondant se met à tourner sur lui-même.

Étapes de validations :

- On commence le test avec le robot dans une position où il a la place de faire un tour autour de lui-même.
- Ensuite grâce au frontend, nous validons la présence du robot sur l'interface
- Nous vérifions que le bouton afin d'identifier est disponible
- Nous pouvons maintenant appuyer sur le bouton qui va lancer l'identification du robot
- Le robot devrait alors faire son mouvement prévu qui est de faire un tour sur lui-même

On réalise ce test au moins 3 fois par robot, pour valider son fonctionnement.

Note : Dans la simulation les mouvements sur l'axe z uniquement fonctionnent mal, la fonction identifier fait donc légèrement avancer le robot en plus de le faire tourner sur lui-même.

#### b. Requis commandes de missions (R.F.2)

On veut vérifier que les robots commencent à se déplacer ou s'arrêtent lorsque l'on démarre ou arrête une mission depuis la page associée. Il doit être possible d'effectuer ces étapes à l'infini, c'est-à-dire que notre serveur et nos robots ne doivent pas se retrouver dans un état instable après le démarrage ou l'arrêt d'une mission.

Étapes de validations :

- Nous positionnons les robots à n'importe quel endroit.
- Nous vérifions que les robots sont en bonnes positions.
- Grâce à l'interface, nous vérifions que les robots sont présents et détectés.
- Nous passons sur le côté de l'interface consacré à la mission.
- Nous pouvons ensuite appuyer sur le bouton de début de mission.
- Il faut alors vérifier quel est le comportement du robot, il devrait adopter le comportement d'exploration.
- Nous vérifions aussi que le statut du robot a changé.
- Nous arrêtons alors la mission.
- Le comportement d'exploration doit alors s'arrêter et le statut du robot doit changer.

Nous relançons puis et arrêtons la mission 4 fois afin de vérifier la résilience de la fonctionnalité.

#### c. Requis états des robots affichés en temps réel dans l'interface (R.F.3)

On veut vérifier que l'état des robots est affiché en temps réel sur notre client.

Étapes de validations :

- On ouvre deux pages de notre site côte à côte.
- La page principale doit récupérer et afficher le nombre de robots allumés ainsi que leur état courant (en mission, en attentes), une case par robot et le nombre de cases sont définis par le nombre de robots connecté (1, 2 ou plus).
- On démarre une mission avec notre premier client, une fois celle-ci démarrer, l'état des robots doit se mettre à jour et passer à « Mission en cours » en 1 seconde maximum. Les boutons **identifier** doivent être désactivés sur notre second client.
- On arrête la mission, l'interface affiche les robots dans l'état « En attente », les boutons **identifier** sont réactivés.

#### d. Requis exploration et évitement d'obstacles (R.F.4 et R.F.5)

Lors d'une mission, les robots doivent explorer l'environnement de manière autonome en évitant les obstacles et les autres robots.


Étapes de validations :

- On démarre une mission, les robots doivent commencer à se déplacer.
- Lors de leur exploration, il faut vérifier que les robots ne touchent ou ne frôlent pas d'obstacles.
- Après quelques minutes on arrête la mission, l'exploration doit s'arrêter pour tous les robots.
- On refait ces étapes plusieurs fois en faisant varier la position des obstacles et la position de départ des robots dans le cas des robots physiques.

#### e. Requis clients en simultanés et support (R.F.10)

On veut vérifier que notre interface web est visualisable pas plusieurs clients en simultané et ce sur au moins deux types de support (ordinateur et tablette).

Étapes de validations :

- On ouvre une première fois notre site et on démarre une mission.
- On ouvre un second client puis en utilisant l'interface de développement (F12) on active la « barre d'outils de l'appareil »  et on sélectionne **iPad Air**, ensuite on navigue vers la page mission pour visualiser la mission en cours.
- Les deux clients doivent pouvoir visualiser les informations sur la mission en cours (logs, carte...) et les interactions faites par l'un des clients doivent être visualisables par tous les clients.

#### f. Affichage de la carte généré par les robots (en cours) (R.F.8)

Nous voulons tester que la carte collectée par la station au sol est précise et ressemble à l'environnement. Il faut aussi s'assurer que la carte se met à jour durant la mission.

Étapes de validations :

- Il faut s'assurer que les robots soient allumés et placés dans la zone qu'ils doivent scanner.
- Nous devons pour ce test commencer une mission.
- Il faut alors basculer sur la vue de mission sur l'interface utilisateur.
- Ici nous pouvons commencer la mission.
- L'utilisateur devrait alors observer une carte peu remplie.
- Nous devons alors vérifier que la carte ressemble à la réalité.
- La carte doit aussi rajouter des détails au cours du temps.
- Nous laissons le temps se passer en vérifiant que la carte se mette à jour de manière logique.
- Nous vérifions que le résultat final est cohérent.

Il faut répéter l'expérience trois fois pour s'assurer de la pérennité des résultats.

#### g. Requis modification du code et mise à jour des robots (R.F.14 et R.F.16)

L'interface web doit permettre de modifier les fichiers des robots indépendamment et de les mettre à jour. Il faut donc pouvoir récupérer la liste des fichiers et leurs contenus, modifier le code depuis un éditeur de texte, renvoyer le fichier modifié à notre robot et enfin recompiler notre code et relancer le système.

Étapes de validations :

- On ouvre notre client et on effectue une action comme **identifier**.
- On navigue vers la page de l'éditeur de code .
- On sélectionne le robot à partir duquel on veut récupérer l'arbre de fichier
- On sélectionne le fichier qui correspond à la fonction identifier (INF3995-Robot/ros\_ws/src/py\_identify\_server/py\_identify\_server/identify.py), une fois celui-ci chargé dans notre éditeur de code on modifie la vitesse de rotation du robot, en augmentant la valeur de `rotate_msg.angular.z`
- On sauvegarde notre fichier en appuyant sur le bouton **Sauvegarder** de l'interface.
- Puis on met à jour via le bouton **Mise à jour** et on attend que le robot redémarre.
- L'exécution de fonction identifier permet alors de visualiser notre modification, le robot tournera plus ou moins vite en fonction de la modification réalisée.
- La mise à jour ne doit pas être possible pendant une mission, on exécute alors les étapes précédentes en démarrant préalablement une mission, il ne doit alors pas être possible de mettre à jour notre robot tant que celle-ci est en cours.

#### h. Enregistrement des données de mission dans la BD (R.F.17- R.F.18)

Les missions génèrent plusieurs données. Dans celles-ci on compte : la carte, les distances parcourues, la durée, l'heure, etc. Ces données sont gardées dans une base de données. Il faut donc vérifier du bon fonctionnement.

Étapes de validations :

- On s'assure d'avoir un robot avec lequel nous pouvons lancer la mission.
- Nous pouvons ensuite lancer une mission et attendre quelque temps.
- Arrêtons alors la mission.
- Ensuite nous prenons en note les différents métas donnés que nous nous attendons à observer plus tard.
- Nous prenons aussi une capture d'écran de la carte.
- Une fois cela fait nous allons dans la page dédiée à l'historique des missions.
- Nous pouvons alors comparer les résultats attendus avec les résultats perçus pour les métadonnées.
- Il faut aussi vérifier que la carte est bien la bonne grâce à la capture d'écran prise.

Nous répétons ce test 3 fois avec un nombre de robots différents afin de vérifier l'intégrité des données.

#### i. Requis affichage des logs et informations envoyées par les robots (R.C.1)

Notre système doit lire les informations écrites par les robots (logs, données des capteurs...), pour les afficher dans notre interface de mission. Le comportement attendu est que lorsqu'une mission est en cours, les messages écrits par les robots dans leur console `/rosout` doivent être envoyés aux utilisateurs présents sur la page mission.

Étapes de validation:

- On lance la simulation ou le robot
- On ouvre notre client.
- À partir de la page mission, on démarre une mission
- À une fréquence de 1hz, les messages écrits par les robots sur le topic `/rosout` doivent apparaître sur l'écran de mission.
- Arrêter la mission, après les logs confirmant l'arrêt de la mission, tout autre log apparaissant sur le robot doit arrêter d'être affiché sur l'écran de la mission.

#### j. Requis générations de murs aléatoires (R.C.3)

À chaque démarrage de la simulation, on doit noter la présence de 4 murs formant un pavé fermé et de 3 obstacles dans cette zone placée différemment à chaque redémarrage de celle-ci.

Étapes de validations :

- On lance la simulation plusieurs fois et regardant que la position des obstacles change.
- Nous devons toujours observer des obstacles apparaître, mais les murs externes doivent rester à la même position.
- Enfin, aucun obstacle ne doit apparaître à l'intérieur d'un robot dans la simulation.