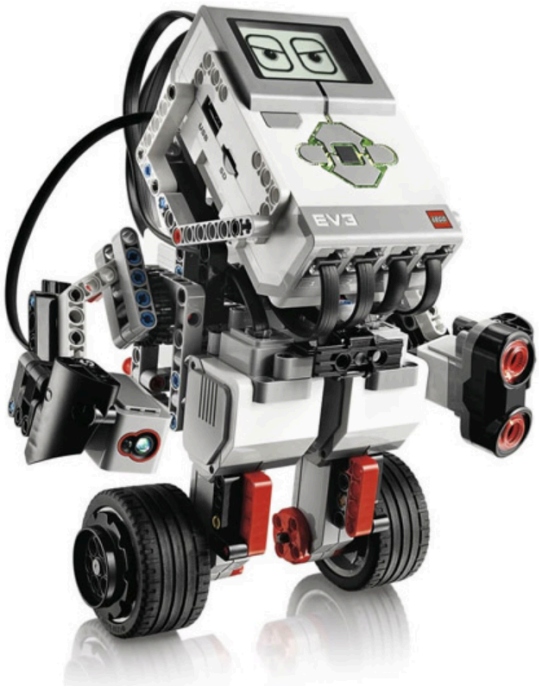
Robot Rmasseur

Plan de Tests



L3 MIASHS
Intelligence Artificielle

AIT EL HADJ Anas
AMACHAT Yousra
HALILY Youssef
NGUIRANE Marième
TAMSOURI Mohammed

Contexte

Dans le cadre de ce projet robotique, nous développons un robot autonome capable de ramasser des palets sur un plateau en un minimum de temps. Ce projet, réalisé sur une période de 12 semaines, vise à programmer un robot utilisant la plateforme LeJOS EV3. L'objectif principal est de concevoir un programme intégré permettant au robot de naviguer efficacement, de localiser et de rassembler des palets tout en respectant les contraintes imposées par la forme prédéterminée du robot, le délai imparti ainsi que la communication entre les membres du groupe. Ce document détaille les différentes phases du plan de test, assurant une validation rigoureuse du logiciel et du matériel intégrés au robot pour répondre aux exigences spécifiées et optimiser les performances lors de la compétition finale.

---

1. Introduction

1.1 Objectifs et Méthodes

Objectifs :

- **Assurer la Qualité** : Garantir que chaque composant du robot fonctionne conformément aux spécifications.
- **Détecter les Anomalies** : Identifier et corriger les dysfonctionnements avant la mise en production.
- **Valider l'Intégration** : S'assurer que les différents modules interagissent de manière harmonieuse et efficace.
- **Optimiser les Performances** : Vérifier que le robot répond aux critères de performance définis (vitesse, précision, réactivité).

Méthodes :

- **Tests Unitaires** : Validation des fonctionnalités individuelles de chaque composant (capteurs, actionneurs).
- **Tests Fonctionnels** : Vérification des scénarios d'utilisation spécifiques (mouvement, rotation, recherche de palet).
- **Tests d'Intégration** : Évaluation de l'interaction entre les différents modules du système.
- **Vérification de la Documentation** : S'assurer que la documentation associée est complète, précise et à jour.

1.2 Documents de Référence

- **Documentation Officielle de LeJOS EV3** : LeJOS EV3 Documentation
- **Cahier des Charges** : Description détaillée des exigences et des spécifications du projet.
- **Plan de Développement** : Organisation, division et gestion des différentes étapes et tâches du projet.
- **Normes de Codage Java** : Standards à suivre pour le développement et la maintenance du code.
- **Guide de Bonnes Pratiques en Test Logiciel** : Méthodologies recommandées pour la réalisation des tests.

---

2. Guide de Lecture

2.1. Maîtrise d'œuvre

Équipe de Développement :

- **AIT EL HADJ Anas**
- **AMACHAT Yousra**
- **HALILY Youssef**
- **NGUIRANE Marième**
- **TAMSOURI Mohammed**

L'équipe de maîtrise d'œuvre est responsable de l'exécution des tâches techniques du projet, incluant la programmation, l'intégration des capteurs et actionneurs, ainsi que la réalisation des tests fonctionnels et unitaires.

2.2. Maîtrise d'ouvrage

Responsable :

- **PELLIER Damien**

La maîtrise d'ouvrage, représentée par PELLIER Damien, est chargée de définir les exigences du projet, de superviser l'avancement, et d'évaluer les résultats finaux pour s'assurer que le robot répond aux objectifs fixés.

3. Concepts de Base

L'objectif du projet est de concevoir et de programmer un robot capable d'exécuter efficacement la tâche consistant à rassembler le plus grand nombre possible de palets dans l'en-but adverse en 3 minutes. La clé réside dans la création d'un programme intégré qui permet au robot de naviguer efficacement, de localiser et de rassembler des palets tout en respectant les contraintes imposées par la forme prédéterminée du robot, la contrainte de délai ainsi que la contrainte de communication entre les membres du groupe. Le projet réalisé par l'équipe sera évalué lors d'une compétition rigoureuse après une période de préparation de 12 semaines. Notre objectif ultime est de remporter la compétition en démontrant la fiabilité, la précision et l'efficacité de notre robot. L'évaluation consistera en une compétition finale ainsi qu'à la soumission de documents décrivant notre méthodologie, notre choix de conception et nos résultats.

4. Tests Fonctionnelles

Les tests fonctionnels visent à vérifier que le robot accomplit correctement les tâches pour lesquelles il a été conçu. Chaque scénario de test est détaillé selon une structure standard pour assurer une couverture exhaustive des fonctionnalités.

4.1. Scénario de Mouvement

4.1.1. Identification

- **Nom du Scénario** : Test de Mouvement de Base
- **Objectif** : Vérifier que le robot peut avancer et reculer de manière précise.

4.1.2. Description

Le robot doit avancer tout droit pour récupérer un palet le plus rapidement possible, puis l'amener dans l’en-but adverse. Ensuite, il doit successivement ramener le maximum de palets restants sur le terrain pour marquer plus de points.

4.1.3. Contraintes

- **Environnement** : Surface plane et dégagée pour éviter les interférences avec les mouvements du robot.
- **Conditions Initiales** : Le robot doit être à l'arrêt et correctement calibré avant le début du test.
- **Sécurité** : Assurer que le robot ne risque pas d'endommager son environnement ou lui-même durant le test.

4.1.4. Dépendances

- **Capteurs de Mouvement** : Les capteurs doivent être fonctionnels et correctement calibrés.
- **Actionneurs de Déplacement** : Les moteurs responsables du mouvement doivent être opérationnels.
- **Alimentation Électrique** : Batterie suffisamment chargée pour exécuter le test sans interruption.

4.1.5. Procédure de Test

1. **Préparation** : Placer le robot sur une surface plane et dégagée.
2. **Commande d'Avancement** : Envoyer la commande pour que le robot avance de 50 cm.
3. **Observation** : Vérifier que le robot avance la distance spécifiée avec précision.
4. **Commande de Recul** : Envoyer la commande pour que le robot recule de 50 cm.
5. **Vérification** : S'assurer que le robot revient à sa position initiale.
6. **Résultats** : Noter les écarts éventuels entre les distances commandées et parcourues.
7. **Méthodes de Test Utilisées** :
    - `avancer(double distance)`
    - `avancerSync(double distance)`
    - `isMoving()`
    - `stop()`

4.2. Scénario de Rotation

4.2.1. Identification

- **Nom du Scénario** : Test de Rotation de 90°
- **Objectif** : Vérifier que le robot peut effectuer une rotation précise de 90 degrés.

4.2.2. Description

Le robot doit tourner de 90 degrés à droite, puis de 90 degrés à gauche, et enfin compléter une rotation de 360 degrés. L'objectif est de s'assurer que le robot peut ajuster sa direction de manière précise et revenir à son orientation initiale après une rotation complète.

4.2.3. Contraintes

- **Espace Suffisant** : Assurer que le robot dispose de suffisamment d'espace pour effectuer les rotations sans obstruction.
- **Conditions de Surface** : Surface plane pour éviter des rotations erronées dues à des irrégularités.
- **Calibration Initiale** : Le robot doit être correctement calibré pour les rotations précises.

4.2.4. Dépendances

- **Actionneurs de Rotation** : Les moteurs responsables de la rotation doivent être opérationnels.
- **Capteurs d'Orientation** : Les capteurs permettant de mesurer l'angle de rotation doivent fonctionner correctement.
- **Logiciel de Contrôle** : Les algorithmes de rotation doivent être exempts de bugs pour assurer des mouvements précis.

4.2.5. Procédure de Test

1. **Préparation** : Positionner le robot de manière à disposer de suffisamment d'espace pour tourner.
2. **Commande de Rotation Droite** : Envoyer la commande pour que le robot tourne de 90° à droite.
3. **Observation** : Vérifier que le robot effectue une rotation précise de 90°.
4. **Commande de Rotation Gauche** : Envoyer la commande pour que le robot tourne de 90° à gauche.
5. **Vérification** : S'assurer que le robot revient à son orientation initiale.
6. **Commande de Rotation Complète** : Envoyer la commande pour que le robot effectue une rotation de 360°.
7. **Observation** : Vérifier que le robot retourne à sa position et orientation initiales.
8. **Résultats** : Noter les écarts éventuels dans les angles de rotation.
9. **Méthodes de Test Utilisées** :
    - `tourner(double angle)`
    - `tournerSync(double angle)`
    - `isMoving()`
    - `stop()`

4.3. Scénario de Recherche et Capture de Palet

4.3.1. Identification

- **Nom du Scénario** : Test de Recherche et Capture de Palet
- **Objectif** : Vérifier que le robot peut détecter et capturer un palet de manière autonome.

4.3.2. Description

Ce test évalue la capacité du robot à détecter un palet à l'aide de ses capteurs ultrason et tactile, à naviguer vers le palet, et à le capturer en utilisant la pince. L'objectif est d'assurer que toutes les étapes de détection, navigation, et capture fonctionnent en synergie pour accomplir la tâche.

4.3.3. Contraintes

- **Position du Palet** : Placer le palet à une distance et une orientation réalistes pour le test.
- **Environnement** : Surface stable et dégagée pour permettre une navigation fluide.
- **Calibration des Capteurs** : Les capteurs doivent être correctement calibrés pour détecter le palet avec précision.

4.3.4. Dépendances

- **Capteurs Ultrason et Tactile** : Doivent être fonctionnels pour la détection du palet.
- **Actionneurs de Déplacement** : Moteurs responsables de la navigation doivent être opérationnels.
- **Pince** : Doit être capable d'ouvrir et de fermer pour saisir le palet.
- **Algorithmes de Détection et Navigation** : Doivent être exempts de bugs pour assurer une détection et une navigation précises.

4.3.5. Procédure de Test

1. **Préparation** : Placer le palet à une distance de 40 cm devant le robot.
2. **Activation du Scénario** : Démarrer le programme de recherche de palet.
3. **Détection** : Observer si le robot détecte le palet à l'aide du capteur ultrason.
4. **Navigation** : Vérifier que le robot se déplace vers le palet de manière autonome.
5. **Capture** : S'assurer que la pince du robot s'ouvre, saisit le palet, et le sécurise.
6. **Vérification** : Confirmer que le palet a été capturé avec succès.
7. **Résultats** : Noter les éventuelles erreurs de détection, de navigation, ou de capture.
8. **Méthodes de Test Utilisées** :
    - `detecterLesObjets()`
    - `bestObjet(ArrayList<float[]> objets)`
    - `attraperPalet(boolean premier)`
    - `versCouleur(String couleurCible)`

---

5. Tests d’Intégration

Les tests d'intégration visent à vérifier l'interaction entre les différents modules du système, s'assurant que les composants fonctionnent ensemble de manière harmonieuse pour accomplir des tâches complexes.

5.1. Scénario d'Intégration : Détection et Capture Intégrée

5.1.1. Identification

- **Nom du Scénario** : Intégration Détection et Capture de Palet

- **Objectif** : Valider que les modules de détection, navigation, et action de capture fonctionnent en synergie.

5.1.2. Description

Ce test intègre les fonctionnalités de détection d'objets, navigation autonome, et capture de palets. Il vérifie que le robot peut détecter un palet, naviguer vers lui, et le capturer sans intervention externe, en utilisant les données des capteurs et les commandes des actionneurs de manière coordonnée.

5.1.3. Contraintes

- **Configuration des Modules** : Assurer que tous les modules sont correctement configurés et initialisés.
- **Environnement de Test** : Doit permettre une navigation fluide et une détection fiable des palets.
- **Synchronisation des Modules** : Les modules doivent être synchronisés pour éviter les conflits et les comportements inattendus.

5.1.4. Dépendances

- **Capteurs** : Ultrason et tactile doivent fonctionner correctement pour la détection.
- **Actionneurs** : Moteurs de déplacement et pince doivent être opérationnels.
- **Logiciel de Contrôle Intégré** : Algorithmes coordonnant la détection, navigation, et capture doivent être exempts de bugs.

5.1.5. Procédure de Test

1. **Préparation** : Configurer le robot avec tous les modules actifs et calibrés.
2. **Placement du Palet** : Positionner un palet à une distance réaliste devant le robot.
3. **Lancement du Scénario** : Démarrer le programme d'intégration.
4. **Observation** : Surveiller le processus de détection, navigation, et capture.
5. **Vérification** : Confirmer que le palet a été détecté, atteint, et capturé sans erreur.
6. **Résultats** : Noter les éventuels problèmes d'intégration, tels que des décalages dans les données des capteurs ou des défaillances dans les actionneurs.
7. **Méthodes de Test Utilisées** :
    - `testDetecterLesObjets()`
    - `testBestObjet()`
    - `testAttraperPalet(boolean premier)`
    - `testVersCouleur(String couleurCible)`

---

6. Tests Unitaires

Les tests unitaires sont destinés à vérifier les fonctionnalités individuelles de chaque composant du système, en isolant chaque module pour s'assurer qu'il fonctionne correctement indépendamment des autres.

6.1. Test Unitaire : Classe Agent

6.1.1. Identification

- **Nom du Test** : TestAgent
- **Objectif** : Valider les méthodes principales de la classe Agent.

6.1.2. Description

Ce test vérifie les principales fonctionnalités de la classe Agent, telles que la détection des objets, la sélection du meilleur objet, la capture des palets et la détection des couleurs.

6.1.3. Contraintes

- **Environnement de Test** : Surface plane avec des objets positionnés à des distances et angles spécifiques.
- **Calibration** : Les capteurs doivent être correctement calibrés avant le test.
- **Conditions de Lumière** : Assurer un éclairage stable pour les tests de détection de couleur.

6.1.4. Dépendances

- **Composants Physiques** : Moteurs, capteurs ultrason, capteurs tactiles et capteurs de couleur doivent être opérationnels.
- **Logiciel de Contrôle** : Les algorithmes de détection et de navigation doivent être exempts de bugs.
- **Alimentation Électrique** : Batterie suffisamment chargée pour exécuter les tests sans interruption.

6.1.5. Procédure de Test

1. **Préparation** : Initialiser le robot et s'assurer que tous les capteurs et actionneurs sont fonctionnels.
2. **Exécution des Tests** :
    - **Test de Détection des Objets** : Exécuter `testDetecterLesObjets()` pour vérifier la détection des objets.
    - **Test de Sélection du Meilleur Objet** : Exécuter `testBestObjet()` pour valider la sélection du meilleur objet.

    - **Test de Capture de Palet** : Exécuter `testAttraperPalet(true)` pour vérifier la capture du premier palet.
    - **Test de Détection de Couleur** : Exécuter `testVersCouleur("White")` pour vérifier la détection de la couleur blanche.
3. **Observation et Vérification** : Noter les résultats et vérifier que chaque méthode fonctionne comme attendu.
4. **Résultats** : Documenter les résultats de chaque test, noter les succès et les échecs éventuels.
5. **Méthodes de Test Utilisées** :
    - `testDetecterLesObjets()`
    - `testBestObjet()`
    - `testAttraperPalet(boolean premier)`
    - `testVersCouleur(String couleurCible)`

6.2. Test Unitaire : Classe CapteurUltrasonTest

6.2.1. Identification

- **Nom du Test** : CapteurUltrasonTest
- **Objectif** : Valider les fonctionnalités du capteur ultrason.

6.2.2. Description

Ce test vérifie que le capteur ultrason mesure correctement les distances, détecte les objets et les palets, et réagit de manière appropriée lors des déplacements et rotations.

6.2.3. Contraintes

- **Environnement de Test** : Surface plane avec des objets à des distances précises.
- **Calibration** : Le capteur ultrason doit être correctement calibré.
- **Conditions de Lumière** : Assurer des conditions de lumière stables pour éviter les interférences avec le capteur.

6.2.4. Dépendances

- **Capteur Ultrason** : Doit être fonctionnel et correctement connecté.
- **Logiciel de Mesure** : Algorithme de mesure de distance doit être précis.
- **Équipement de Mesure** : Outil de mesure pour valider les distances réelles.

6.2.5. Procédure de Test

1. **Préparation** : Initialiser le capteur ultrason et placer des objets à différentes distances.
2. **Exécution des Tests** :

    - **Test Fonctionnel** : Exécuter `testFonction()` pour vérifier la détection d'objets et de palets.
    - **Test de Distance** : Exécuter `getDistanceTest(25.0f)` pour valider la précision de la mesure à 25 cm.
    - **Test de Détection d'Objet** : Exécuter `detectObjectTest(70.0f)` pour vérifier la détection lors du déplacement.
    - **Test de Détection de Palet** : Exécuter `detecterPaletTest()` pour valider la détection de palets lors des rotations.
3. **Observation et Vérification** : Noter les distances mesurées, les objets détectés et les palets capturés.
4. **Résultats** : Documenter les résultats de chaque test, identifier les écarts et les corriger.
5. **Méthodes de Test Utilisées** :
    - `testFonction()`
    - `getDistanceTest(float expectedDistance)`
    - `detectObjectTest(float distance)`
    - `detecterPaletTest()`

6.3. Test Unitaire : Classe Test_deplacement

6.3.1. Identification

- **Nom du Test** : Test_deplacement
- **Objectif** : Valider les fonctions de déplacement du robot.

6.3.2. Description

Ce test vérifie que les méthodes de déplacement du robot fonctionnent correctement, notamment les rotations, les avancées synchrones et asynchrones, et l'état de mouvement du robot.

6.3.3. Contraintes

- **Espace Suffisant** : Assurer que le robot dispose de suffisamment d'espace pour effectuer les mouvements et rotations.
- **Surface Plane** : Surface stable pour éviter des mouvements imprécis.
- **Calibration Initiale** : Le robot doit être correctement calibré pour les déplacements et rotations précises.

6.3.4. Dépendances

- **Actionneurs de Déplacement** : Les moteurs responsables des déplacements et rotations doivent être opérationnels.

- **Capteurs d'Orientation** : Les capteurs utilisés pour mesurer l'angle de rotation doivent fonctionner correctement.
- **Logiciel de Contrôle** : Les algorithmes de déplacement doivent être précis et exempts de bugs.

6.3.5. Procédure de Test

1. **Préparation** : Initialiser le système de déplacement du robot.
2. **Exécution des Tests** :
    - **Test de Rotation** : Exécuter `test_getDirection_tourner()` pour vérifier la précision des rotations.
    - **Test d'Avancement et d'État de Mouvement** : Exécuter `test_avancerSync_avancer_isMoving()` pour valider les déplacements synchrones et asynchrones.
    - **Test de Rotation Asynchrone** : Exécuter `test_tournerAsync_tourner()` pour vérifier les rotations asynchrones et le rétablissement de la direction initiale.
3. **Observation et Vérification** : Surveiller les mouvements, rotations et vérifier que l'état de mouvement correspond aux attentes.
4. **Résultats** : Documenter les résultats de chaque test, noter les écarts et ajuster les paramètres si nécessaire.
5. **Méthodes de Test Utilisées** :
    - `test_getDirection_tourner()`
    - `test_avancerSync_avancer_isMoving()`
    - `test_tournerAsync_tourner()`
    -

9. Références

- **Documentation Officielle de LeJOS EV3** : LeJOS EV3 Documentation
- **Guide de JUnit 5** : JUnit 5 User Guide
- **Normes de Codage Java** : Java Code Conventions
- **Articles sur les Tests Logiciels et la Robotique** :
    - *"Effective Unit Testing for Robotics Applications"*
    - *"Best Practices for Testing Embedded Systems"*
- **Livres de Référence** :
    - *"Robot Operating System (ROS) for Absolute Beginners"* par Lentin Joseph
    - *"Java: The Complete Reference"* par Herbert Schildt