

# **DOCUMENT COMPLET – PROJET DOFBOT**

**Approche par DÉFIS D'INGÉNIERIE (Agile)**

## **PARTIE A — MOUVEMENTS (ACTION ROBOTIQUE)**

### **DÉFI A1 — MOUVEMENTS ÉLÉMENTAIRES (Bas niveau)**

#### **1. Problème réel**

*Le DOFBOT est présent, mais ses articulations ne sont pas maîtrisées. Chaque mouvement peut être dangereux ou imprécis.*

**Défi : Comment commander chaque articulation individuellement de manière sûre, répétable et contrôlée ?**

#### **2. Objectif attendu**

- Chaque articulation répond aux commandes
- Limites physiques respectées
- Poses de référence définies (home, rest, safe)

#### **3. Travail Team ROS**

- Créer un node ROS de commande articulaire
- Implémenter limites d'angle et vitesse
- Définir poses standards
- Publier/recevoir sur un topic ROS

#### **4. Travail Team IA**

- Observer les capacités physiques
- Documenter les limites et contraintes
- Aucune action directe

#### **5. 20 % critiques**

- Connaissance mécanique réelle
- Publication / lecture d'un message ROS
- Test répétitif lent et fiable

#### **6. Erreurs classiques**

- Bouger plusieurs joints en même temps
- Ignorer les limites physiques
- Tester à pleine vitesse
- Confondre simulation et réel

## 7. Outils existants

- Drivers DOFBOT officiels
- `ros_control`
- Scripts Python de base

## 8. Lien avec le prochain défi

- Préparer le passage à des **trajectoires coordonnées** multi-joints

# DÉFI A2 — MOUVEMENTS COORDONNÉS (Trajectoires)

## 1. Problème réel

*Le bras bouge, mais les mouvements sont saccadés et indépendants.*

**Défi :** Comment synchroniser plusieurs articulations pour obtenir un mouvement fluide et cohérent ?

## 2. Objectif attendu

- Trajectoires fluides et synchronisées
- Répétables et fiables
- Commande encapsulée dans des primitives simples

## 3. Travail Team ROS

- Implémenter interpolation linéaire / cubique
- Synchroniser positions et temps
- Node de trajectoire haut niveau

## 4. Travail Team IA

- Comprendre comment déclencher une action
- Préparer les primitives pour la suite (Pick/Place)

## 5. 20 % critiques

- Interpolation stable
- Timing correct
- Test répété des trajectoires

## 6. Erreurs classiques

- Commander les joints indépendamment
- Ignorer le temps et la vitesse
- Trajectoires non reproductibles

## 7. Outils existants

- `trajectory_msgs/JointTrajectory`
- MoveIt Task Constructor (optionnel)

## 8. Lien avec le prochain défi

- Préparer le robot à exécuter des actions complètes (Pick & Place)

# DÉFI A3 — BOUGER AVEC UNE INTENTION (Pick & Place sans vision)

## 1. Problème réel

*Le robot peut bouger, mais il ne fait rien d'utile.*

**Défi :** Comment transformer une suite de mouvements en action significative ?

## 2. Objectif attendu

- Pick & Place simple et fiable
- Encapsulation dans fonctions ou services ROS (`pick()`, `place()`)

## 3. Travail Team ROS

- Définir primitives d'action : approche, descente, fermeture gripper, levée, déplacement, ouverture
- Gérer timings et vitesses
- Encapsulation dans node `manipulation_actions`

## 4. Travail Team IA

- Comprendre le vocabulaire d'action
- Documenter contraintes et préconditions

## 5. 20 % critiques

- Découper une action en phases simples
- Tester chaque phase isolément
- Répéter et stabiliser

## 6. Erreurs classiques

- Mélanger vision et mouvement trop tôt
- Actions trop complexes

- Absence de gestion d'échec

## 7. Outils existants

- MoveIt Task Constructor
- FSM basiques
- Scripts Pick & Place simples

## 8. Lien avec B3

- Préparer le robot à **agir sur des objets détectés** avec coordonnées réelles

# PARTIE B — ANALYSE & RECONNAISSANCE D'IMAGES

## DÉFI B1 — ACQUISITION & PIPELINE IMAGE

### 1. Problème réel

*Le robot est aveugle.*

**Défi :** Comment fournir un flux image stable et exploitable en temps réel ?

### 2. Objectif attendu

- Image propre, stabilisée, prétraitée
- Topic ROS exploitable

### 3. Travail Team IA

- Capture caméra
- Prétraitement : resize, normalisation, filtrage
- Publication /camera/image\_processed

### 4. Travail Team ROS

- Intégrer le node dans la chaîne ROS
- Gérer paramètres temps réel

### 5. 20 % critiques

- Image stable > modèle complexe
- Pipeline rapide
- Prétraitement simple mais cohérent

### 6. Erreurs classiques

- Trop de filtres
- Latence excessive
- Image mal alignée

## 7. Outils existants

- OpenCV
- cv\_bridge
- sensor\_msgs/Image

## 8. Lien avec B2

- Base pour la reconnaissance d'objets

# DÉFI B2 — RECONNAISSANCE D'OBJETS

## 1. Problème réel

*Le robot voit des pixels mais ne comprend pas ce qu'il voit.*

**Défi : Comment transformer l'image en informations exploitables ?**

## 2. Objectif attendu

- Objet identifié
- Score de confiance
- Résultat stable et structuré

## 3. Travail Team IA

- Intégrer modèle IA existant (YOLO, MobileNet, Detectron)
- Gérer faux positifs
- Publier /detected\_object

## 4. Travail Team ROS

- Souscription au topic
- Visualisation et logging
- Vérification compatibilité avec Pick & Place

## 5. 20 % critiques

- Dataset correct
- Seuils de confiance appropriés
- Stabilité de la détection

## 6. Erreurs classiques

- Lier directement IA et mouvement
- Surentraîner le modèle

- Ignorer les faux positifs

## 7. Outils existants

- TensorRT / ONNX
- Nodes ROS existants pour détection

## 8. Lien avec B3

- Base pour localiser réellement l'objet dans l'espace

# DÉFI B3 — LOCALISATION SPATIALE DE L'OBJET

## 1. Problème réel

*Reconnaitre un objet ne suffit pas, il faut savoir où il est dans l'espace réel.*

**Défi :** Transformer les pixels en coordonnées exploitables par le bras.

## 2. Objectif attendu

- Pose dans le repère `base_link`
- Exploitable directement par le Pick & Place

## 3. Travail Team IA

- Conversion 2D → 3D
- Utilisation de la calibration caméra
- Publication `/object_pose` (`geometry_msgs/PoseStamped`)

## 4. Travail Team ROS

- Vérification cinématique
- Transformation TF
- Validation d'atteignabilité

## 5. 20 % critiques

- Calibration caméra
- TF corrects
- Référentiels cohérents

## 6. Erreurs classiques

- Travaux sans TF
- Hypothèses incorrectes sur caméra
- Ignorer l'erreur de position

## 7. Outils existants

- TF2
- ArUco / Depth Camera
- image\_geometry

## 8. Lien avec C

- Prérequis pour que le robot **agisse intelligemment**

# PARTIE C — DÉCISION & AUTONOMIE

## DÉFI C — FAIRE DES CHOIX (Decision Making)

### 1. Problème réel

*Le robot voit et sait agir, mais ne sait pas quoi faire.*

**Défi :** Introduire une logique de décision claire et robuste.

### 2. Objectif attendu

- Choix d'action autonome
- Gestion des échecs
- Comportement explicable

### 3. Travail Team IA

- Règles simples ou policy IA
- Émission de commandes abstraites (PICK object\_A, IGNORE, RETRY)

### 4. Travail Team ROS

- Machine à états / Behavior Tree
- Validation et sécurisation des commandes
- Gestion des échecs et logs

### 5. 20 % critiques

- FSM simple et lisible
- États explicites
- Logging clair

### 6. Erreurs classiques

- Décision implicite
- IA qui commande directement les moteurs
- États cachés

### 7. Outils existants

- SMACH / Behavior Trees
- ROS2 Lifecycle nodes

## 8. Lien avec A/B

- Fusion de perception et action pour autonomie complète

# CONCLUSION

Ces **9 défis** structurent le projet DOFBOT en **phases logiques et progressives** :

1. A1 – Contrôle articulaire
2. A2 – Trajectoires coordonnées
3. A3 – Actions Pick & Place
4. B1 – Pipeline image
5. B2 – Reconnaissance
6. B3 – Localisation spatiale
7. C – Décision et autonomie

Chaque défi résolu = **une étape démontrable**, exploitable pour un **diagramme de Gantt**, avec dépendances claires et interfaces définies.