



Comment sont programmés les robots ?

Pratiques actuelles et tendances en
recherche

Robin Passama
Ingénieur de recherche CNRS
LIRMM, Université de Montpellier
Département Robotique

Plan

- **Introduction**
- Principes de base
- Architecture logicielle
- Nouvelles tendances
- Conclusion

Contexte

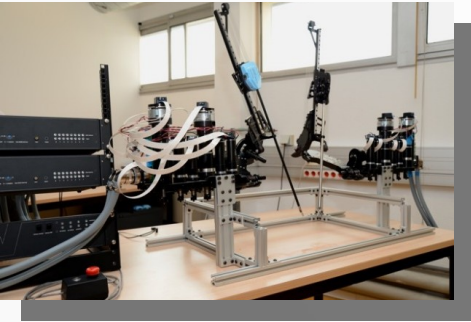
- Qu'est-ce qu'un robot ?

Machine capable d'interagir avec l'environnement physique, disposant d'un niveau d'autonomie donné lui permettant d'exécuter automatiquement des opérations.

- Niveau d'autonomie= capacités **décisionnelles** + ou - grandes
- Interaction : exercer des **forces** sur l'environnement

Contexte

- Quelques exemples de robots



chirurgical



Bras série



cobot



humanoïde



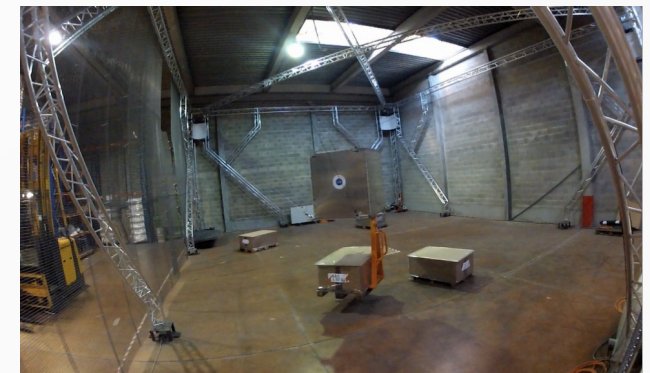
bras parallèles



sous-marin



Mobile
terrestre



Grand robot à câbles

Au niveau matériel

- Des actionneurs



Credits : axesindustries.com

Moteurs électriques



Credits : humarobotics.com

Pompes à vide

- Des capteurs



Credits : directindustry.fr

Encodeurs



Credits : e-motionsupply.com

Capteurs d'efforts

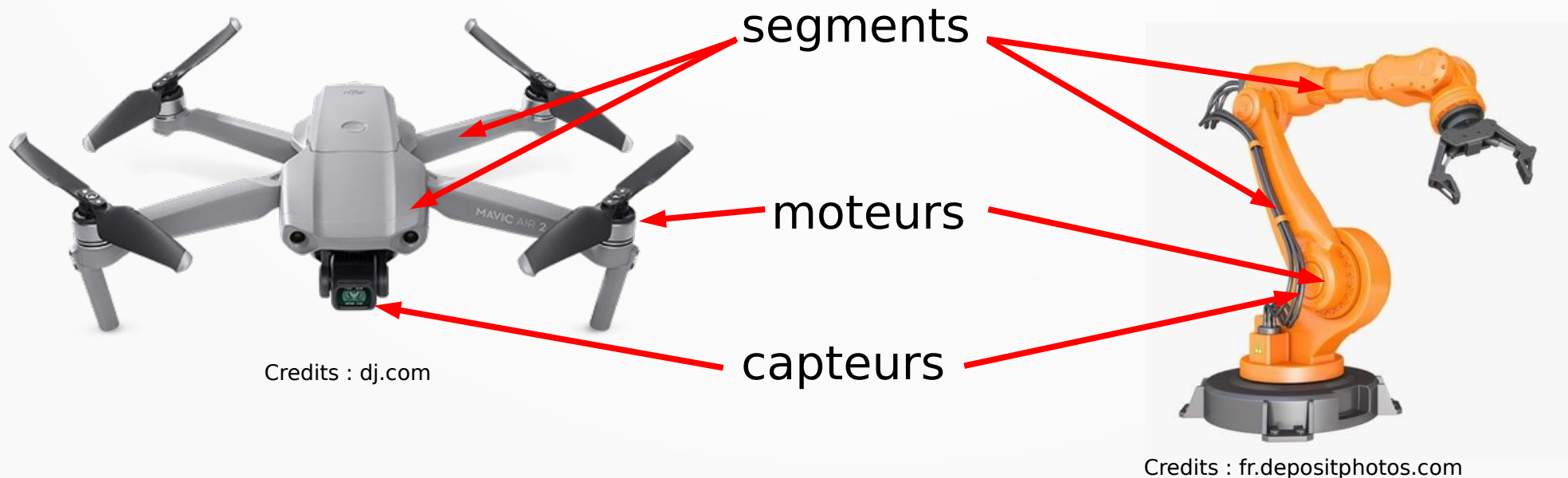


Credits : usinenouvelle.com

Caméras

Au niveau matériel

- Une structure mécatronique :
 - Mécanique : segments, capteurs, actionneurs
 - Electronique : alimentation et communication avec les actionneurs et capteurs ; unités de calcul



En recherche

- Mécanique / Électronique :
 - meilleures performances « physiques » : charrier plus de poids, plus rapide, plus robuste, plus agile, plus précis, moins énergivore, etc.

En recherche

- Informatique / Automatique :
 - plus grande autonomie, meilleures performances décisionnelles

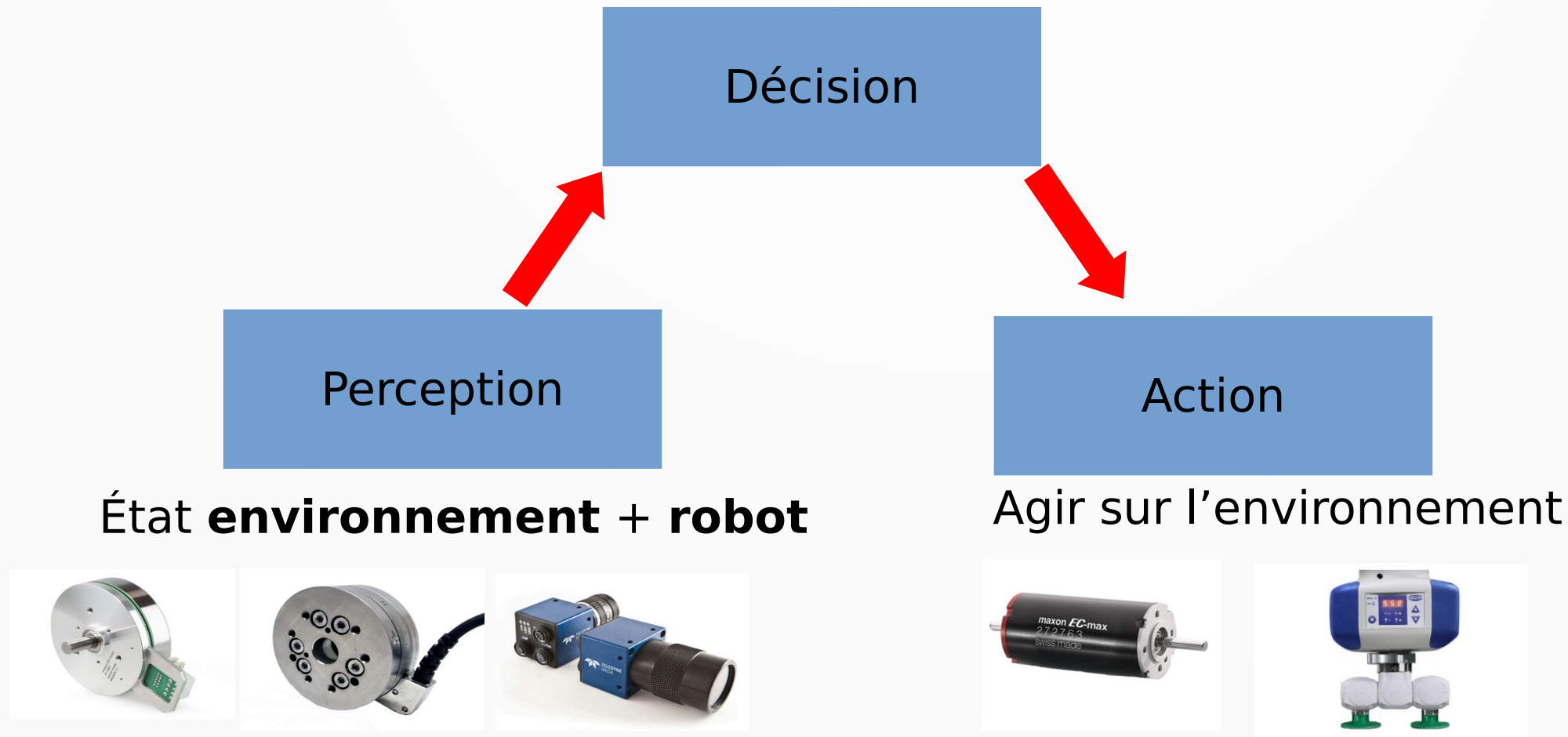


Plan

- Introduction
- **Principes de base**
- Architecture logicielle
- Nouvelles tendances
- Conclusion

Principes de base

- La boucle ***Perception-Décision-Action***

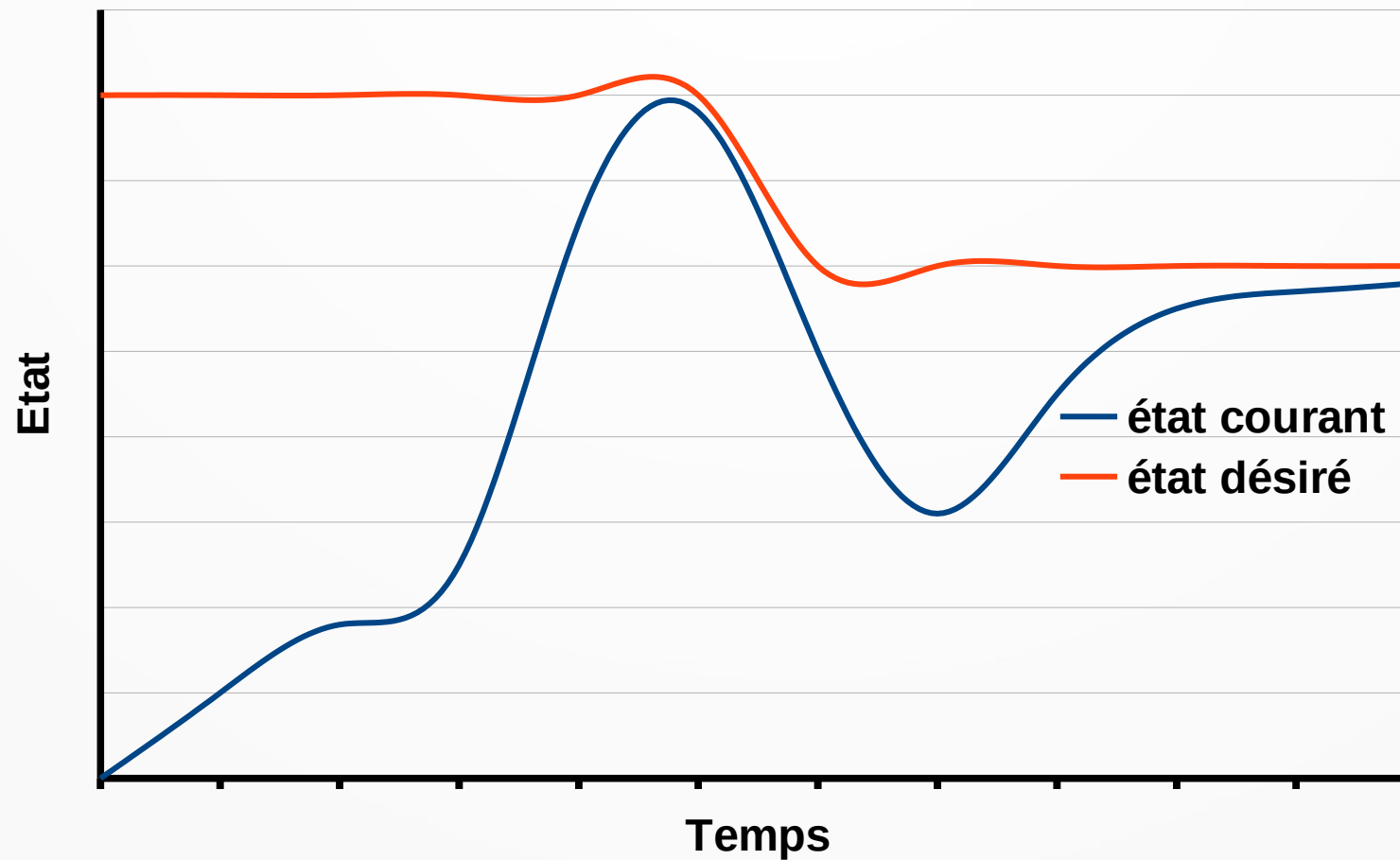


Principes de base

- Programmation **récurrente**
 - Boucle de rétroaction : en fonction de l'**état courant/passé/prédit** piloter les actionneurs de manière à atteindre l'**état désiré**.
 - Exécution **cyclique** et **temps-réel**
 - Contrôler le robot **en continu** pour qu'il réagisse en **temps voulu** aux phénomènes

Principes de base

- Comportement général



Principes de base

- En pratique : structures en **couches**

Vision environnement



Décision

Gestion efforts externes



Décision

Contrôle moteurs



Décision



Principes de base

- Plusieurs couches :
 - Décomposer la prise de décision
 - tout prendre en compte en même temps = difficile
 - Gérer les différentes fréquences d'acquisition :
 - fréquence caméra \ll fréquence encodeur
 - Gérer la dynamique des phénomènes
 - Dynamique de la variation des efforts \gg dynamique de la variation des vitesses

Principes de base

- Au plus près du matériel
 - Exécution **périodique**
 - contraintes **temps-réel fortes**

```
Period p(1ms) ;  
for(;;){  
    auto state = read_sensors();  
    auto cmd = decide(state);  
    control_actuators(cmd);  
    Period.sleep();  
}
```

Principes de base

- Aux plus haut niveaux
 - Exécution **événementielle ou périodique**
 - contraintes **temps-réel** plus faibles

```
Plan plan;

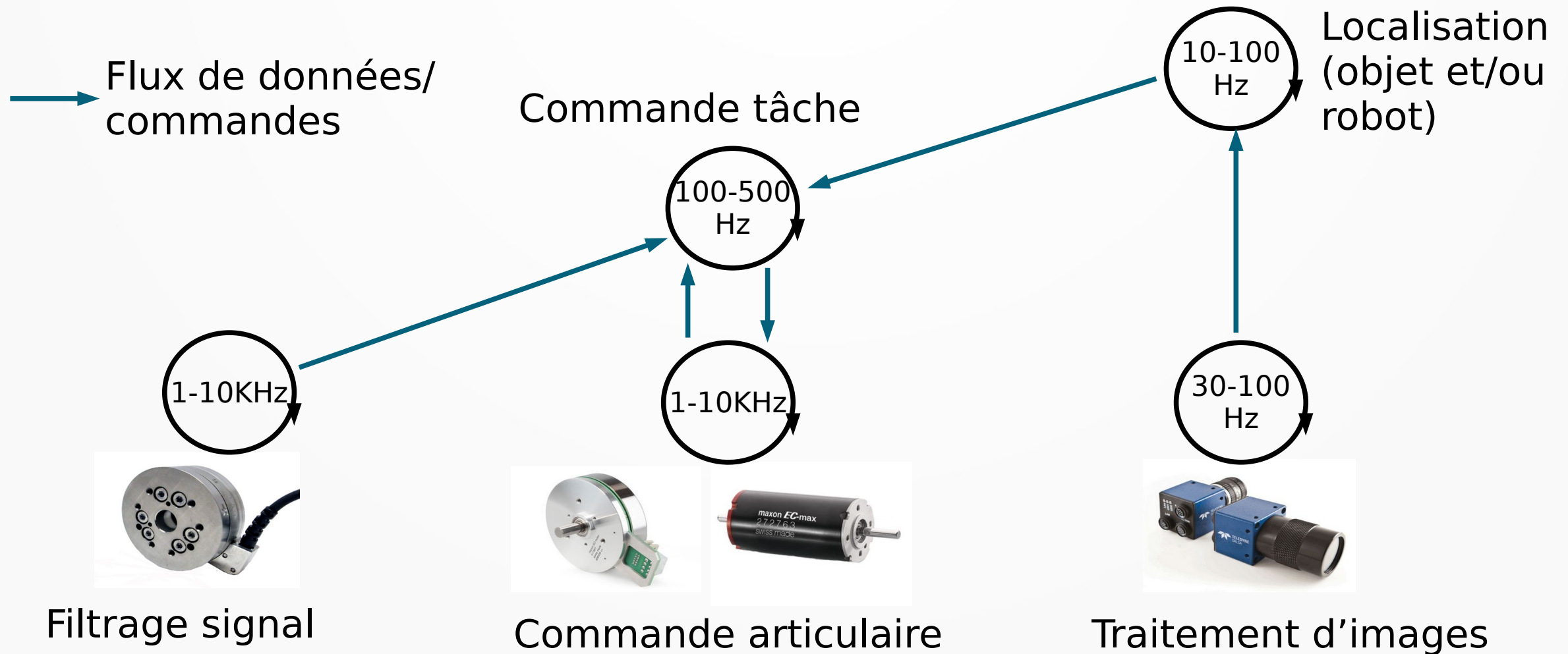
for(auto& action : plan){
    execute_action(action);
    while(auto evt=wait_event()){
        if(action_done(evt)){
            break;
        }
    }
}
```

Plan

- Introduction
- Principes de base
- **Architecture logicielle**
- Nouvelles tendances
- Conclusion

Architecture logicielle

- Vision générale : programmation concurrente

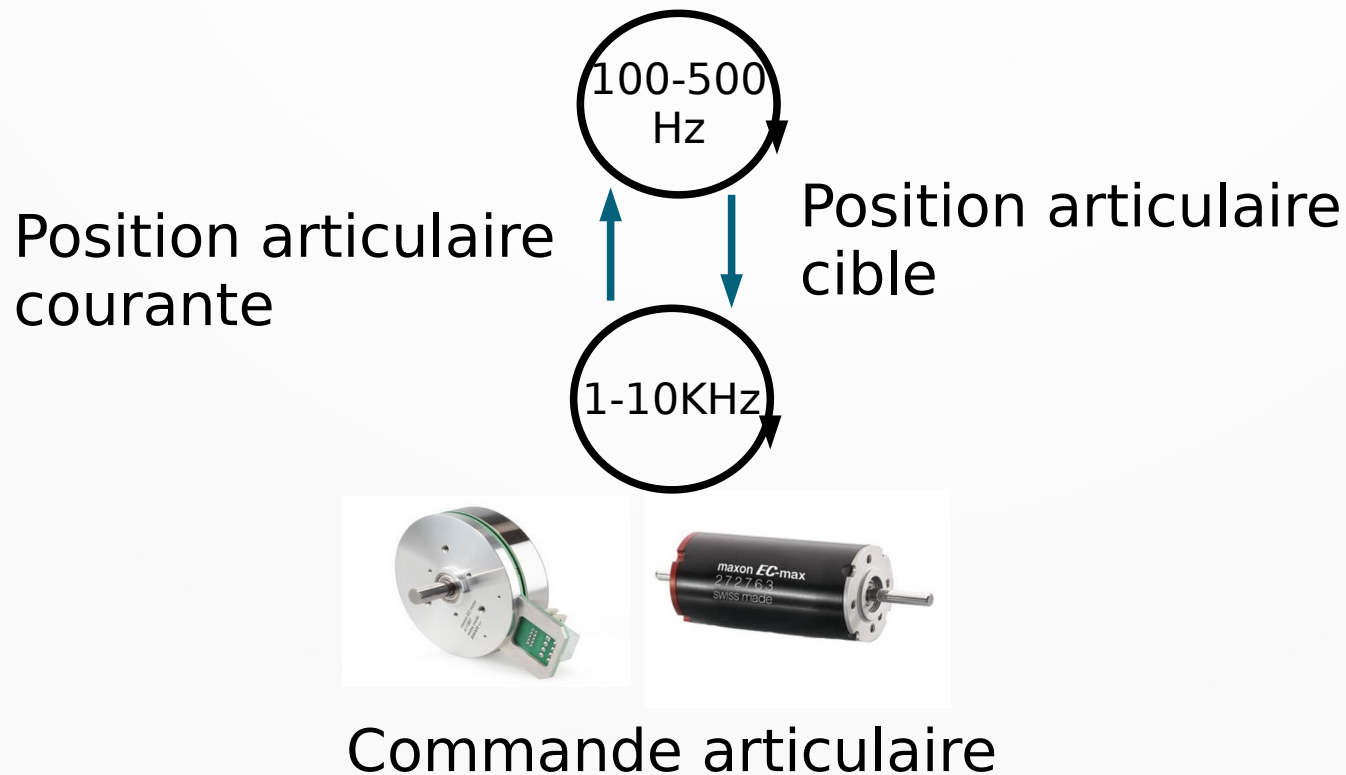


Architecture logicielle

- Langages de programmation :
 - C/C++ : temps-réel, multitâches, drivers, algorithmes efficaces
 - Matlab/Python : prototypage algos, outils « haut niveau » non TR
- Middleware : ROS/ROS2 (<https://www.ros.org/>)
 - Composition « late binding »: Publish/Subscribe (données), RPC (flux de contrôle synchrone/asynchrone)
 - Multi processus / Multi threads
 - Communication « transparente » : DDS (ROS2)
 - Nombreux outils: visualisation, simulation, logging, enregistrement /rejeu, etc.

Architecture logicielle

- Commande dans l'espace articulaire



Commande dans l'espace articulaire

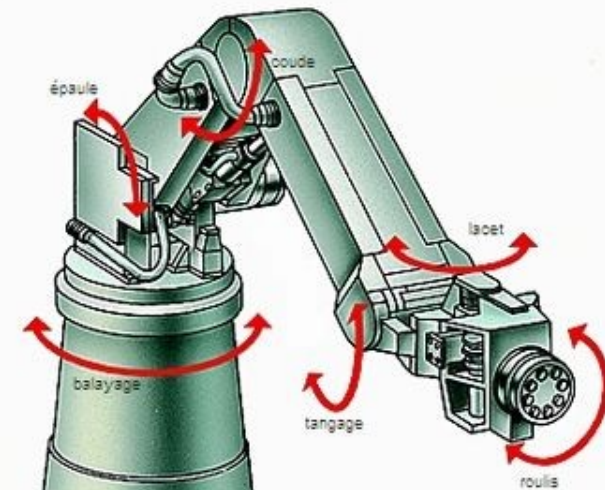
- Contrôler toutes les articulations/moteurs **en même temps**
- Pas de notion géométrique

Change au cours du temps

$$q_d(t) = [q_1(t), \dots, q_n(t)]$$

Position désirée
dans l'espace
articulaire

Vecteur des positions
désirées pour chaque
articulation



Crédits : larousse.fr

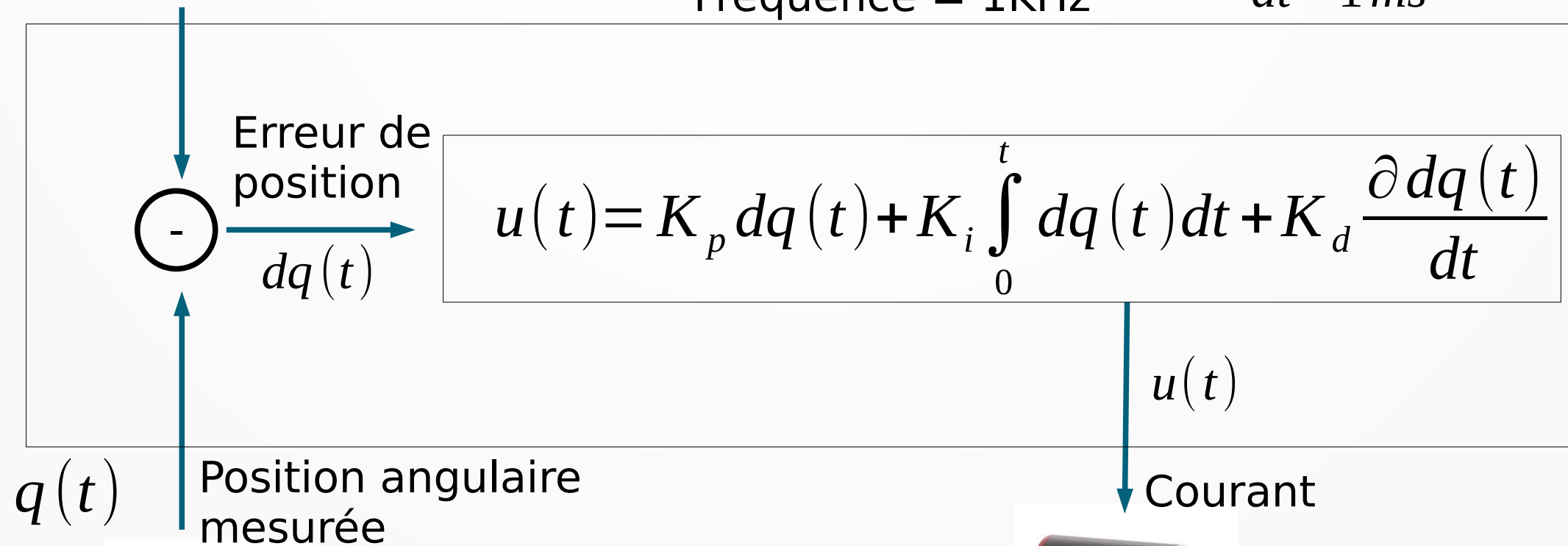
Commande dans l'espace articulaire

- Commande moteur : exemple du correcteur PID

$q_d(t)$ Position angulaire
désirée

Fréquence = 1KHz

$dt = 1\text{ ms}$



Commande dans l'espace articulaire

- Commande moteur : exemple du correcteur PID
 - Enjeu : régler les gains K_p , K_i , K_d

Gain du terme proportionnel

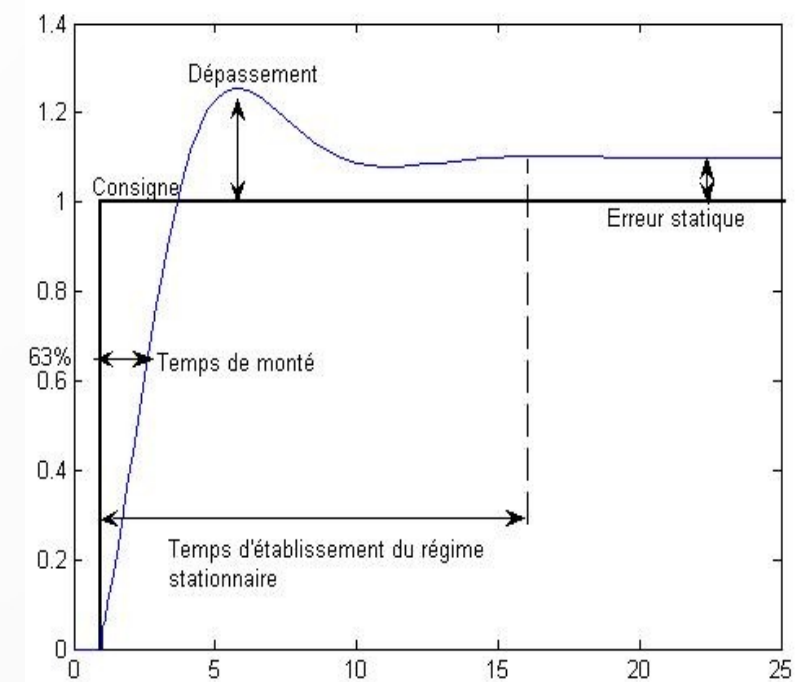
Gain du terme dérivé

$$u(t) = K_p dq(t) + K_i \int_0^t dq(t) dt + K_d \frac{\partial dq(t)}{\partial t}$$

Gain du terme intégral

Accumulation de l'erreur

Vitesse d'accroissement de l'erreur



Crédits : wikipedia.org

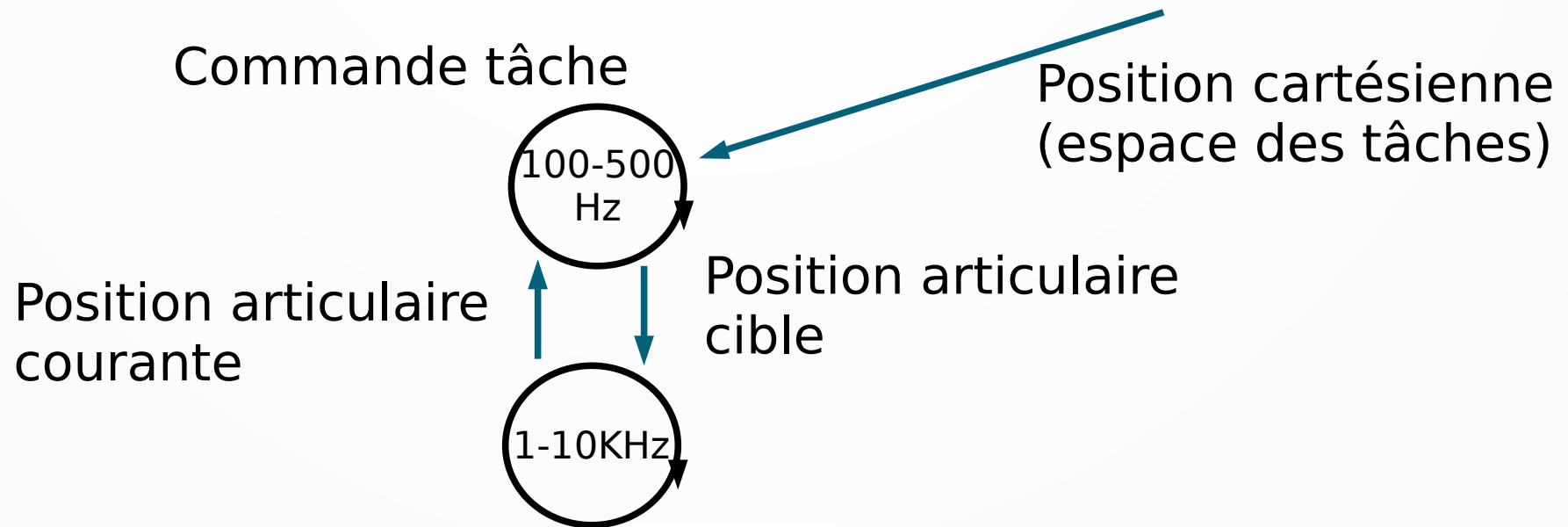
Commande dans l'espace articulaire

- Commande moteur : exemple du correcteur PID

```
Period p(1ms);
double prev_error[NB_MOTORS];
double acc_error[NB_MOTORS];
double cmd_motors[NB_MOTORS];
for(;;) {
    auto curr_state = read_sensors();
    auto des_state = read_desired();
    auto error = des_state - curr_state; //compute error vector
    prev_state_vector=curr_state_vector;
    acc_error += error;
    auto derived_error = error-prev_error;
    auto cmd = Kp*error+Ki*acc_error+Kd*derived_error;
    command_motors(cmd);
    prev_error=error ;
    p.sleep();
}
```

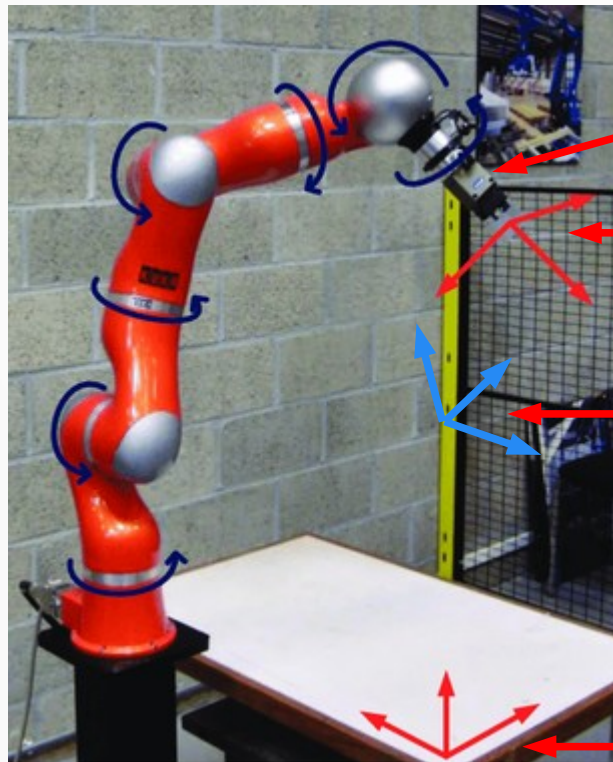
Architecture logicielle

- Commande dans l'espace des tâches



Commande dans l'espace des tâches

- Contrôler les segments du robot dans un (ou+) repère(s) donné(s)



Credits : researchgate.net

Effecteur terminal

Pose courante

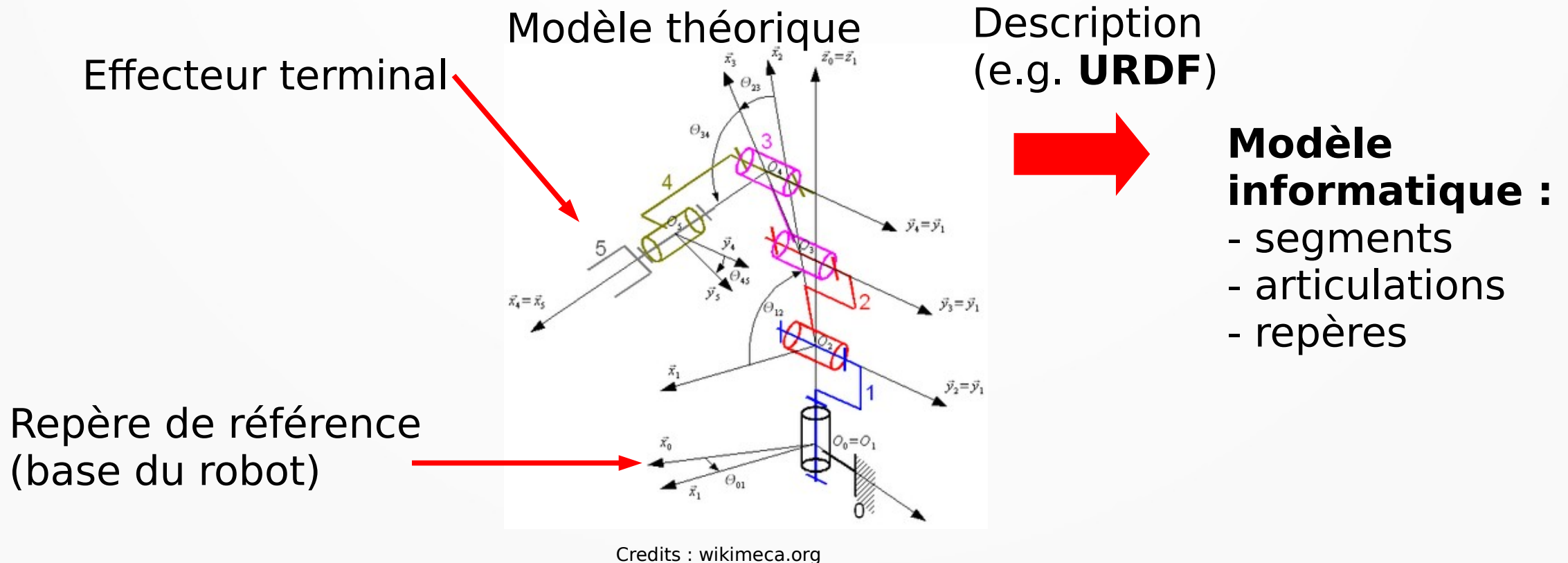
Pose désirée

Exprimée dans

Repère de référence

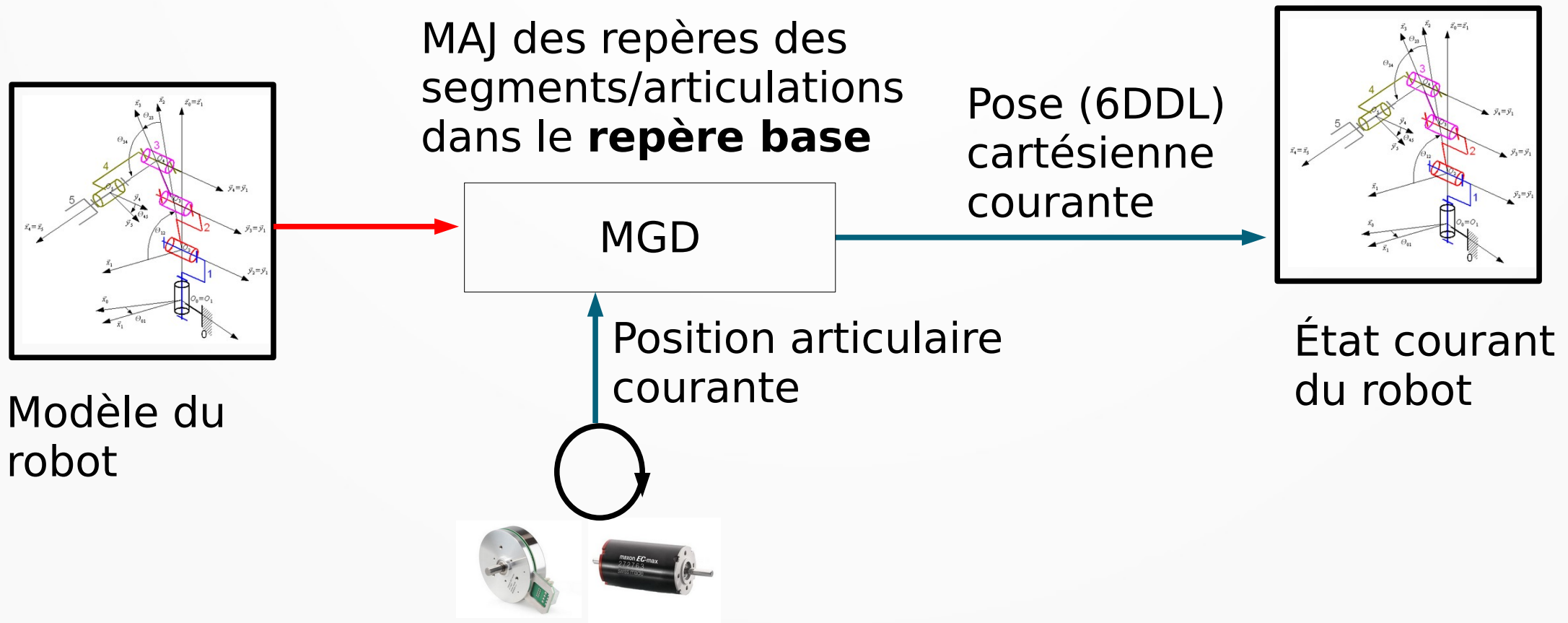
Commande dans l'espace des tâches

- Prise en compte de la géométrie du robot
 - Besoin du modèle du robot



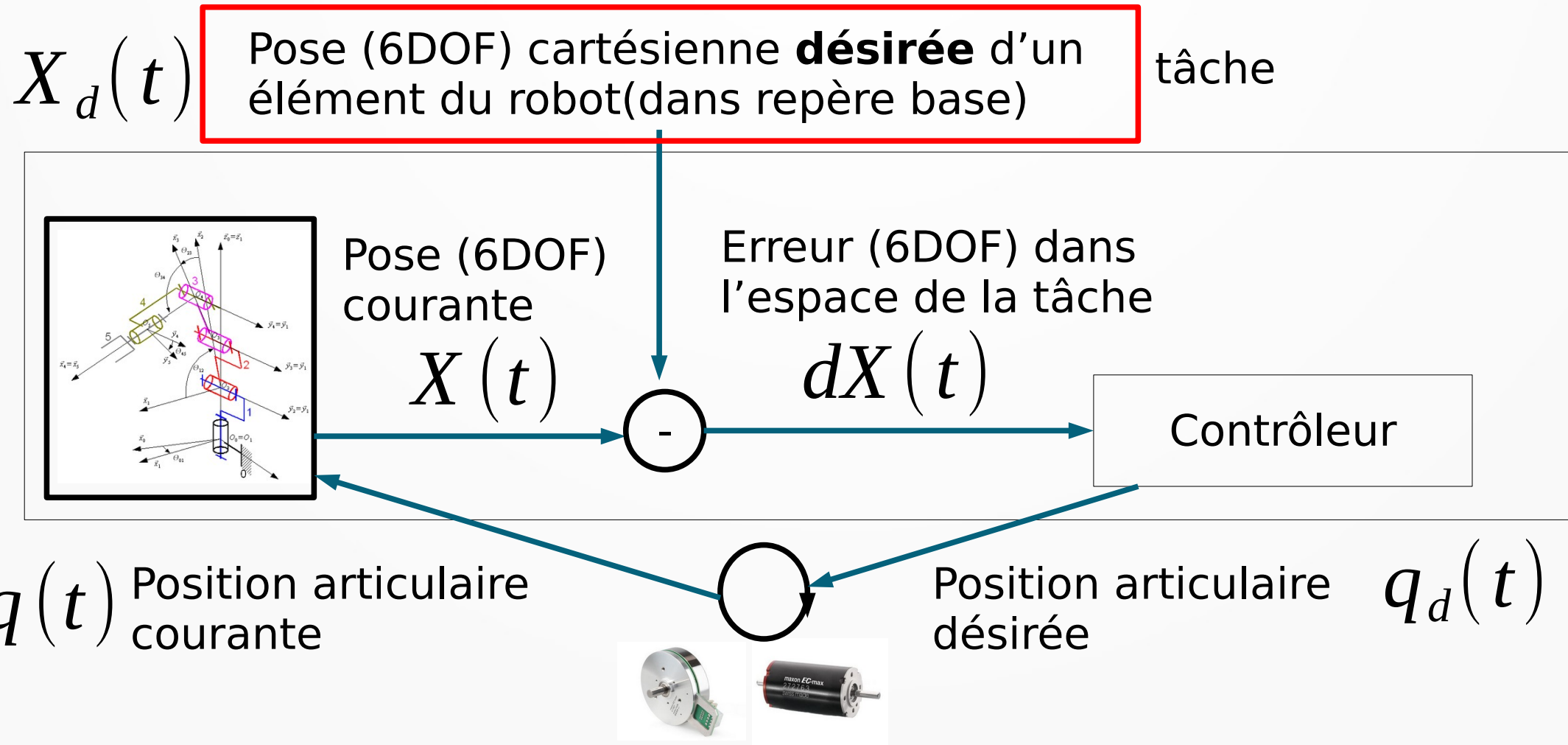
Commande dans l'espace des tâches

- Exploiter le modèle à l'exécution
 - Modèle Géométrique Direct



Commande dans l'espace des tâches

- Commande dans l'espace des tâches



Commande dans l'espace des tâches

- Exploiter le modèle à l'exécution

Modèle Cinématique Direct

$$dX = J(q) dq$$

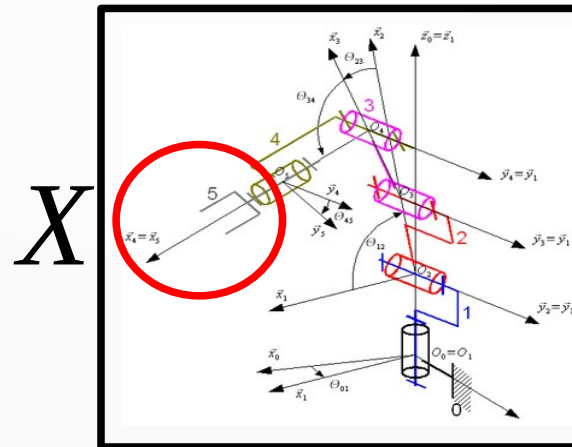
Matrice Jacobienne

$$J(q) = \frac{\partial X}{\partial q}$$

Modèle Cinématique Inverse

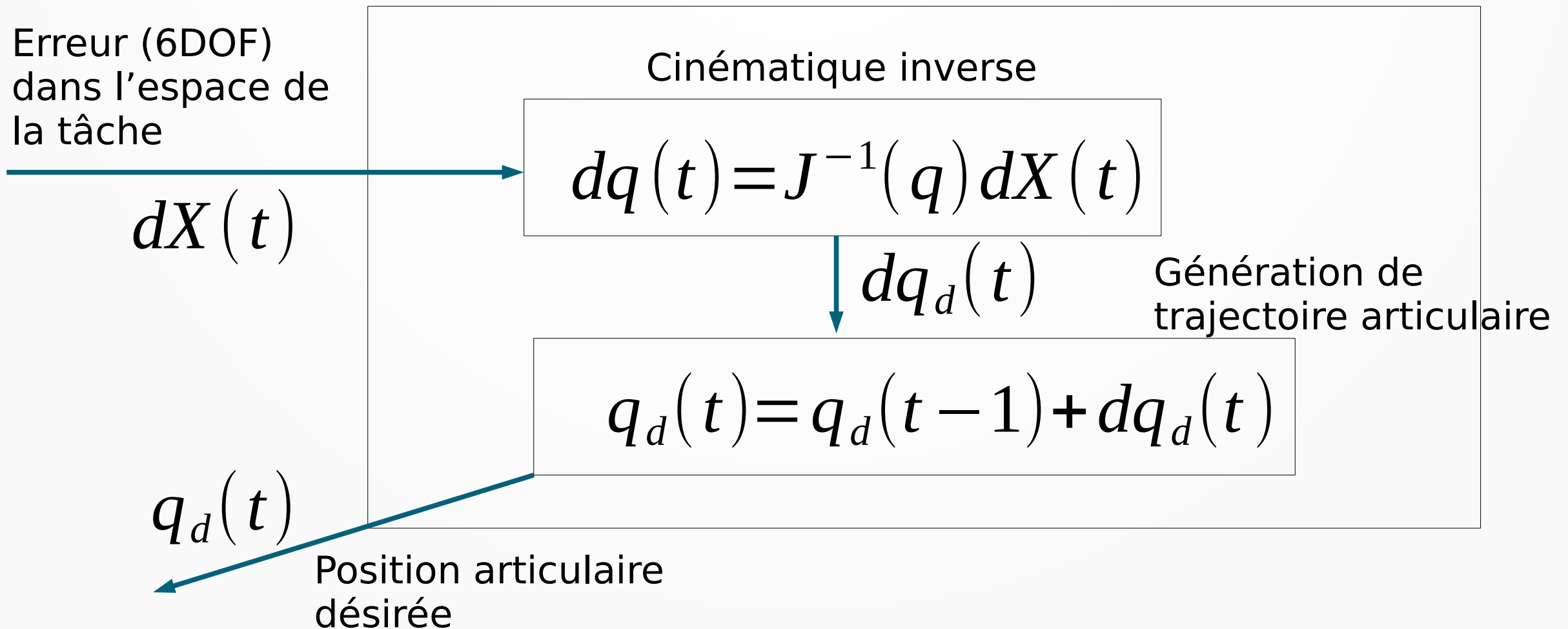
$$dq = J^{-1}(q) dX$$

(pseudo) inverse de la
matrice jacobienne



Commande dans l'espace des tâches

- Contrôleur basé sur la cinématique inverse

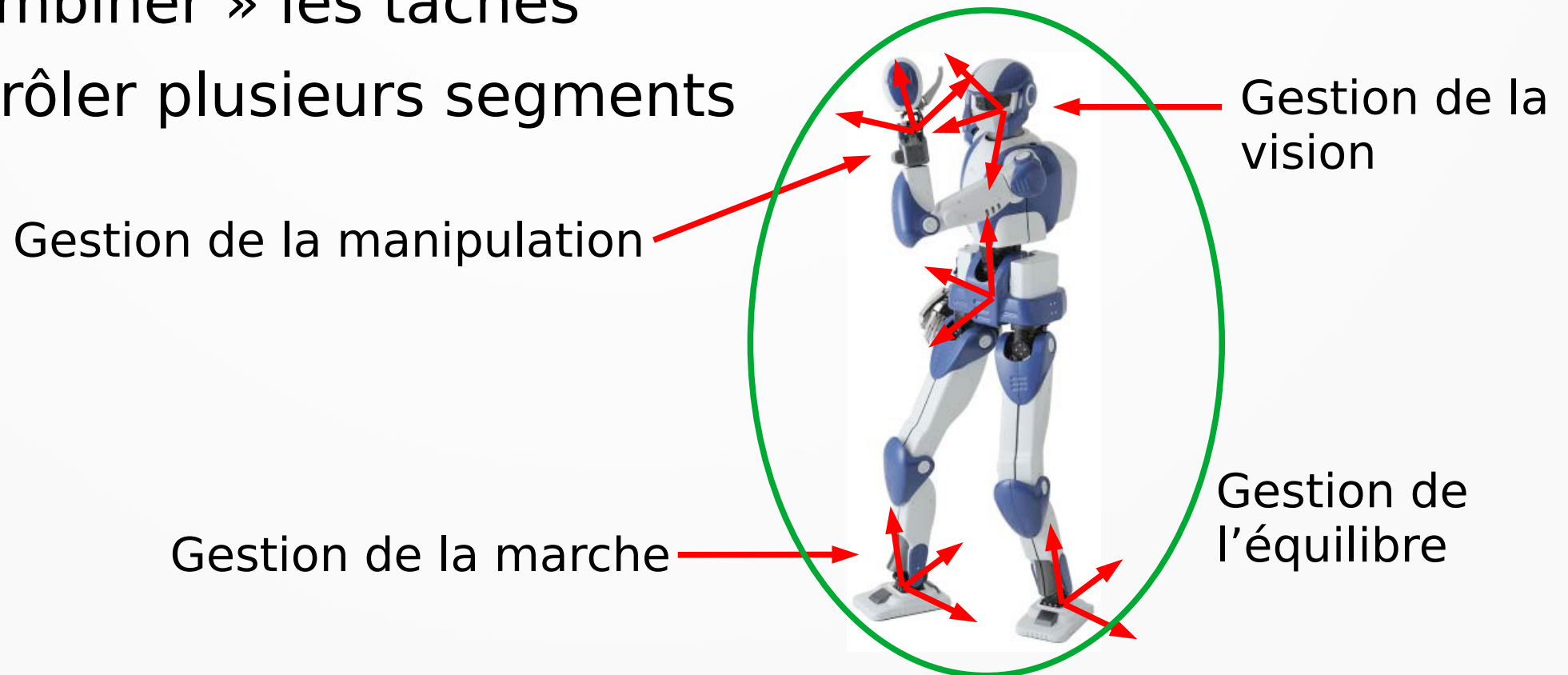


Commande dans l'espace des tâches

- Réification du modèle informatique du robot:
 - Calcul automatique des Jacobiennes
 - Calcul automatique du modèle géométrique direct
 - Extensible à la dynamique directe / inverse
- Limitation : manque de généricité
 - Une tâche (ou un ensemble de tâches) prédéfini par contrôleur
 - Multitâche difficile et spécifique au contrôleur
 - Prise en compte des contraintes (butés, auto-collisions) difficile et spécifique au contrôleur

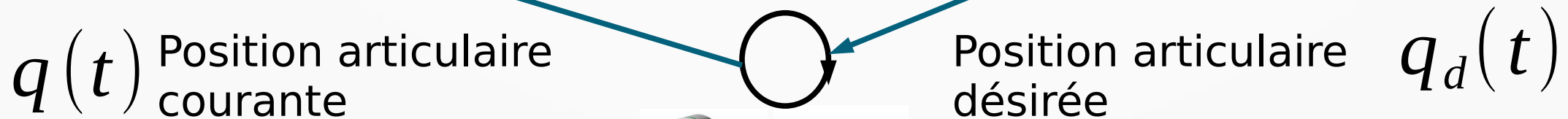
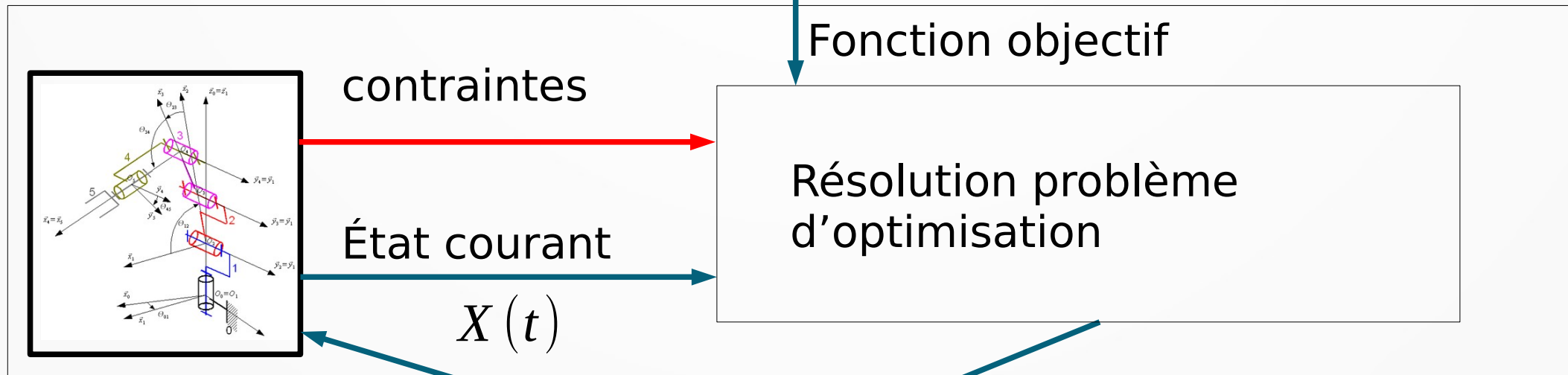
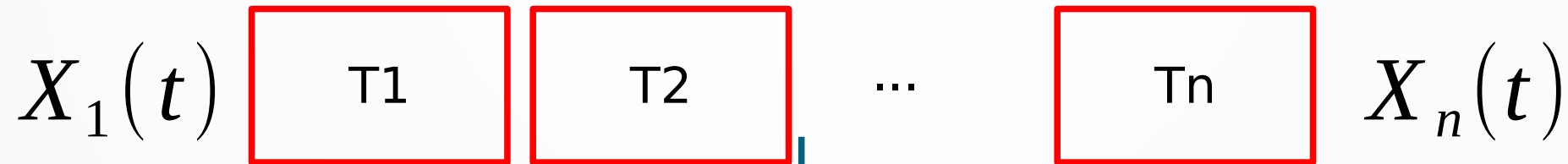
Commande dans l'espace des tâches

- Contrôleurs génériques
 - Permettre de facilement changer de tâche
 - « Combiner » les tâches
 - Contrôler plusieurs segments



Commande dans l'espace des tâches

- Contrôleurs génériques

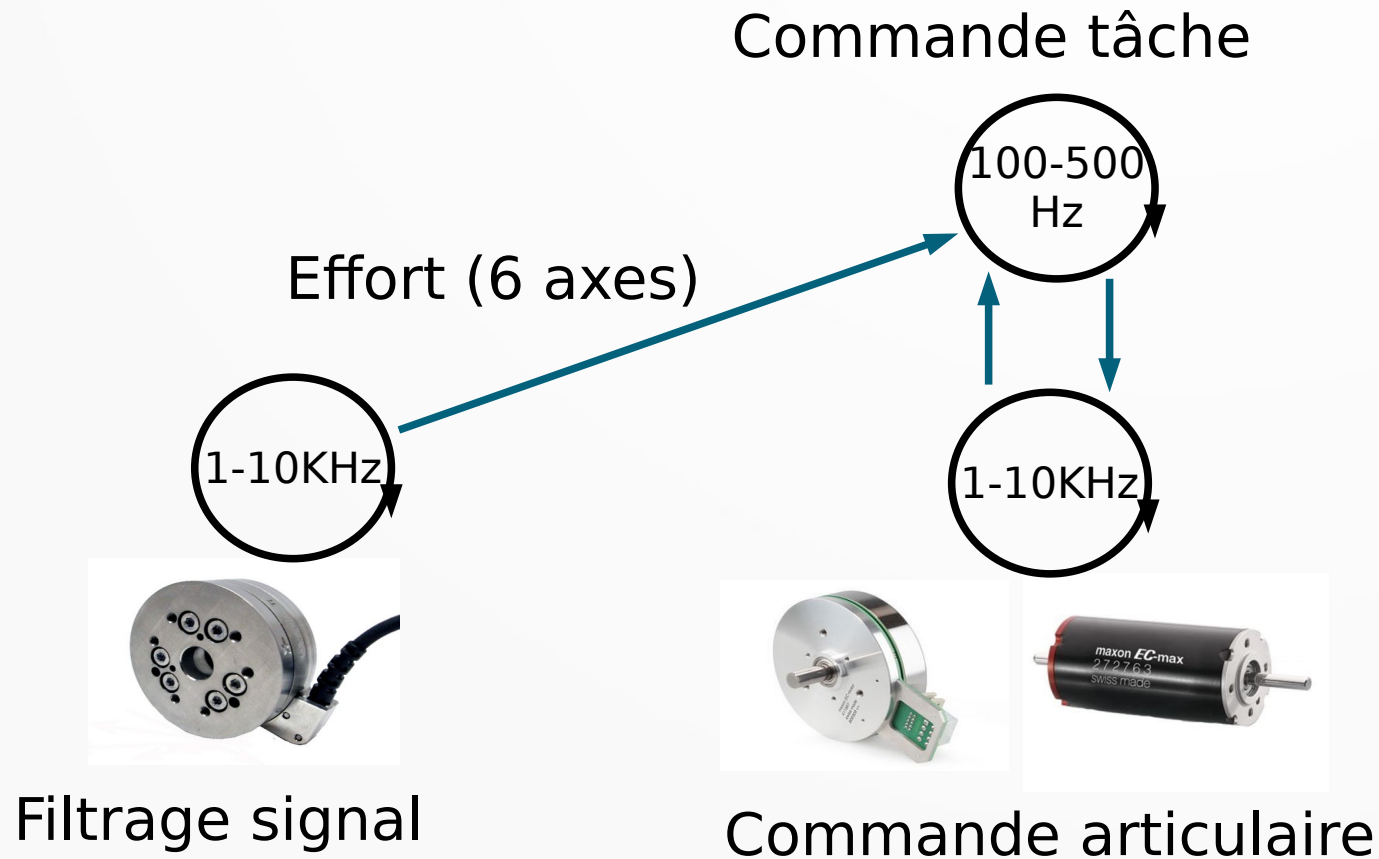


Commande dans l'espace des tâches

- Contrôleurs génériques

Architecture logicielle

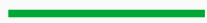
- Filtrage de signaux



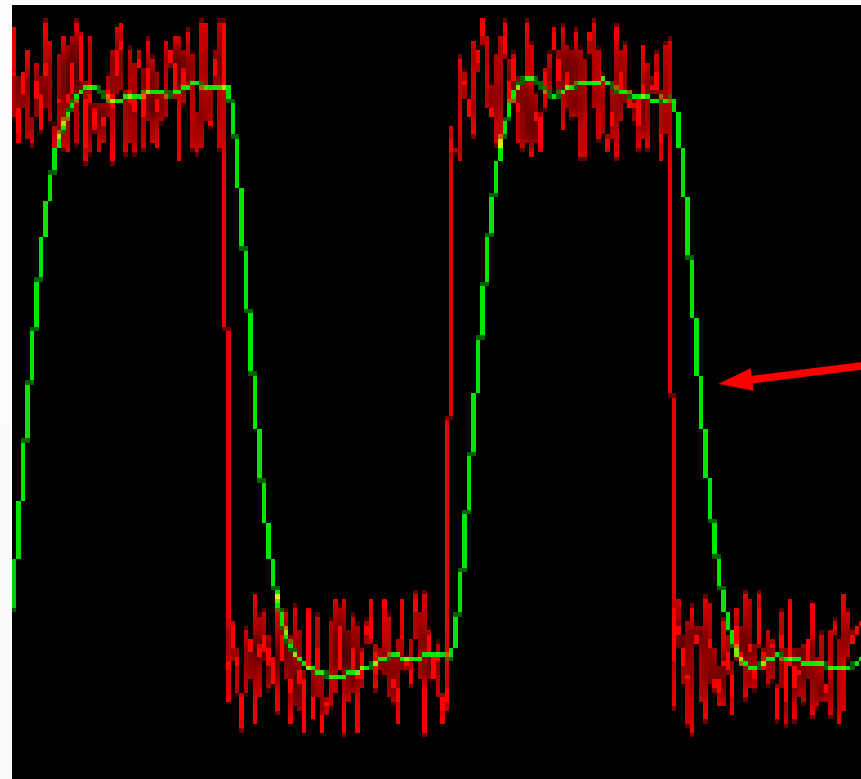
Traitement du signal

- Filtrer le bruit des signaux d'entrée
 - Exemple : filtre passe-bas (1 axe du capteur d'effort)

Signal filtré



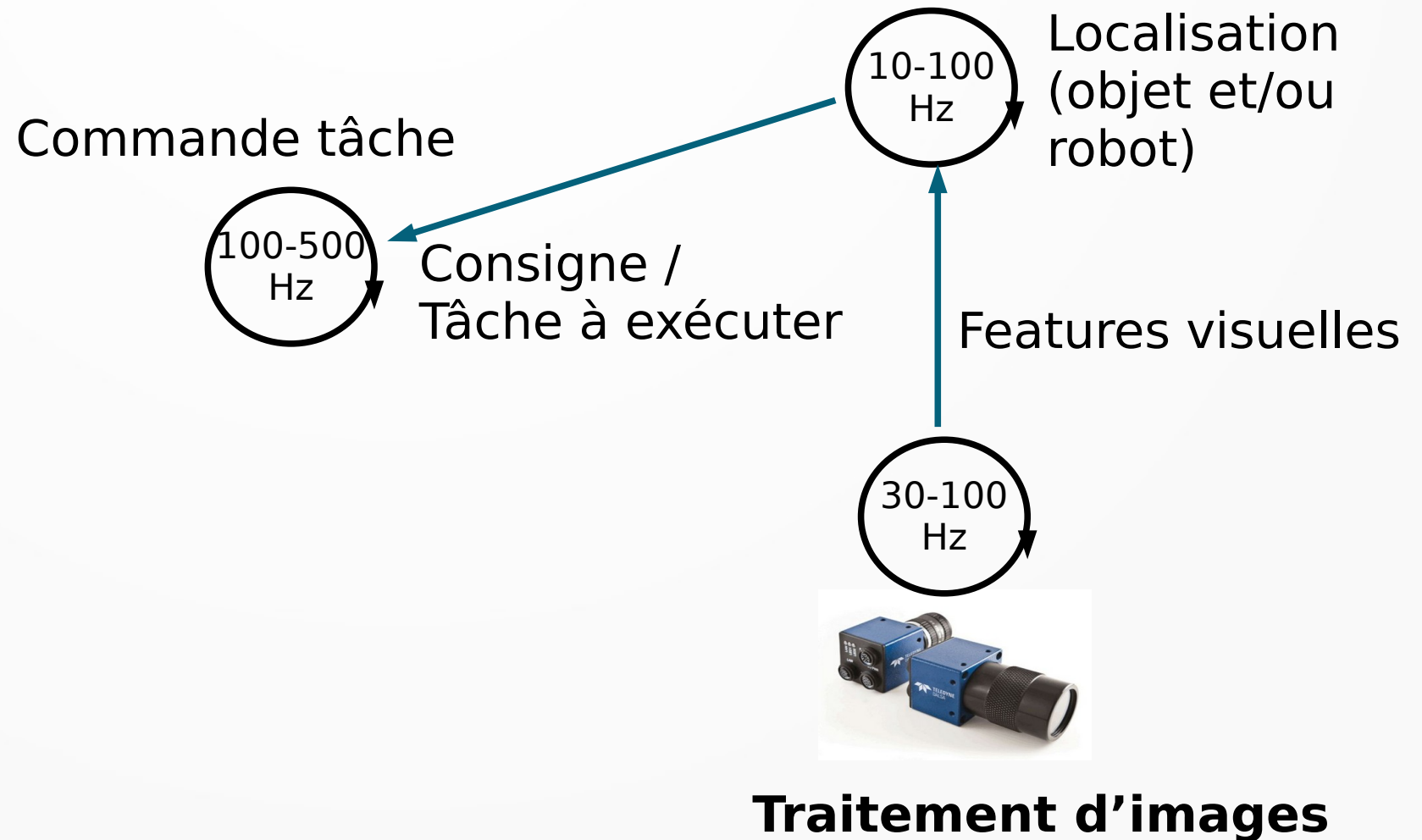
Signal brut



Mesure
exploitable dans
le contrôleur

Architecture logicielle

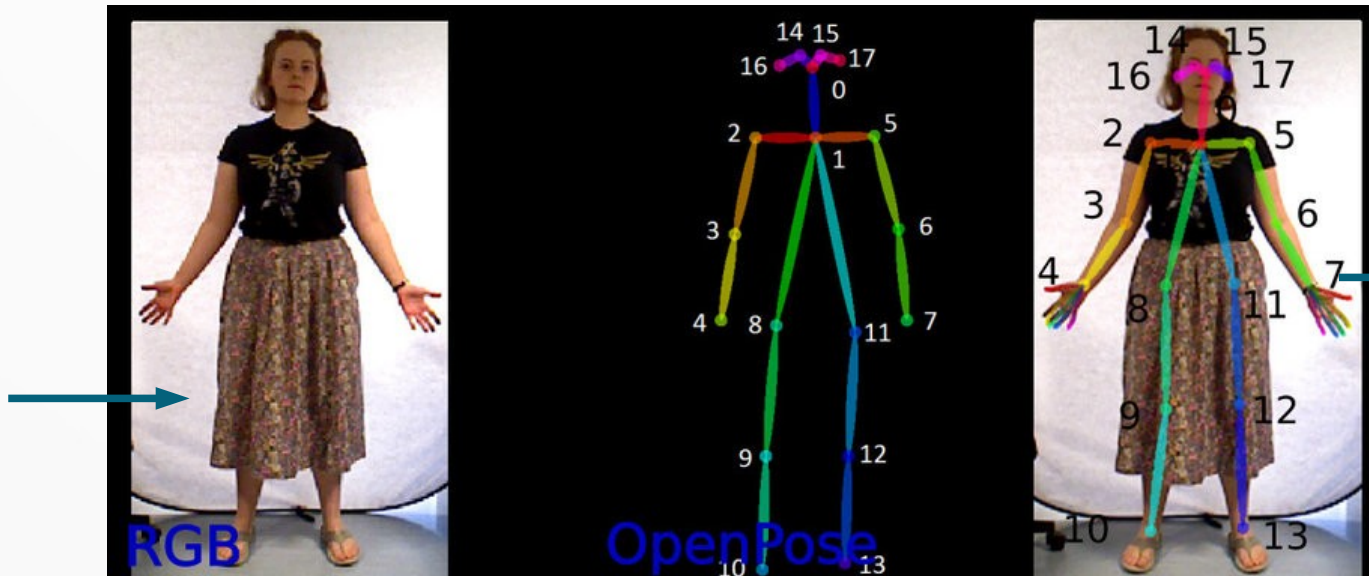
- Perception de l'environnement



Perception de l'environnement

- Traitement d'image
 - Détection d'objets, de personnes, de points d'intérêts, etc.
 - Deep learning (CNN): de plus en plus utilisé

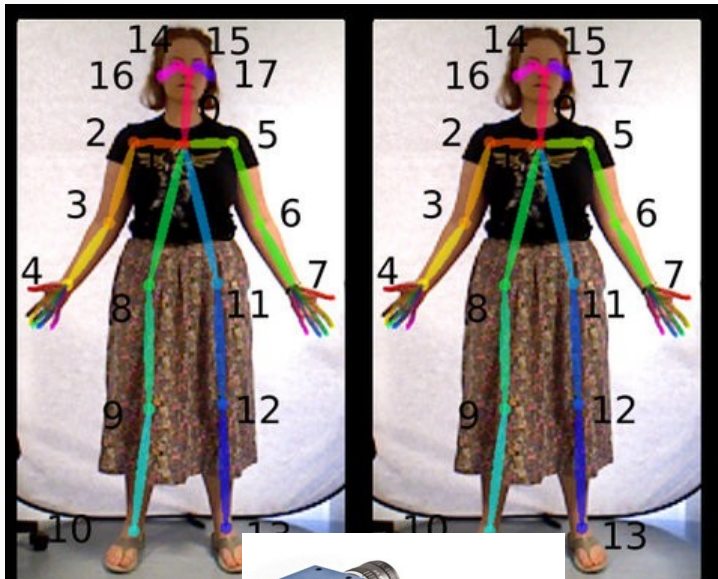
acquisition



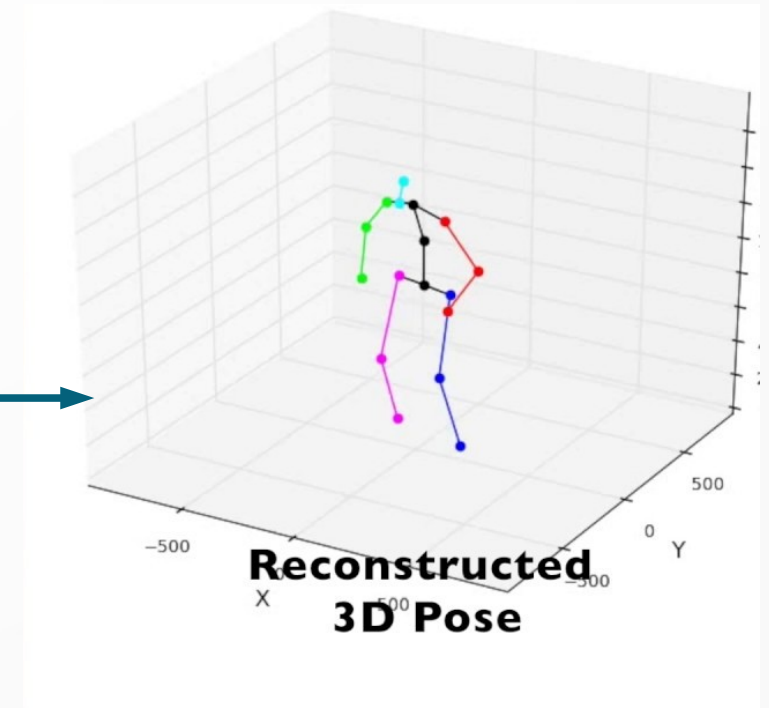
Position 2D
des
articulations

Architecture logicielle

- Localisation 3D de la personne
 - repère image → repère robot



Stereo vision

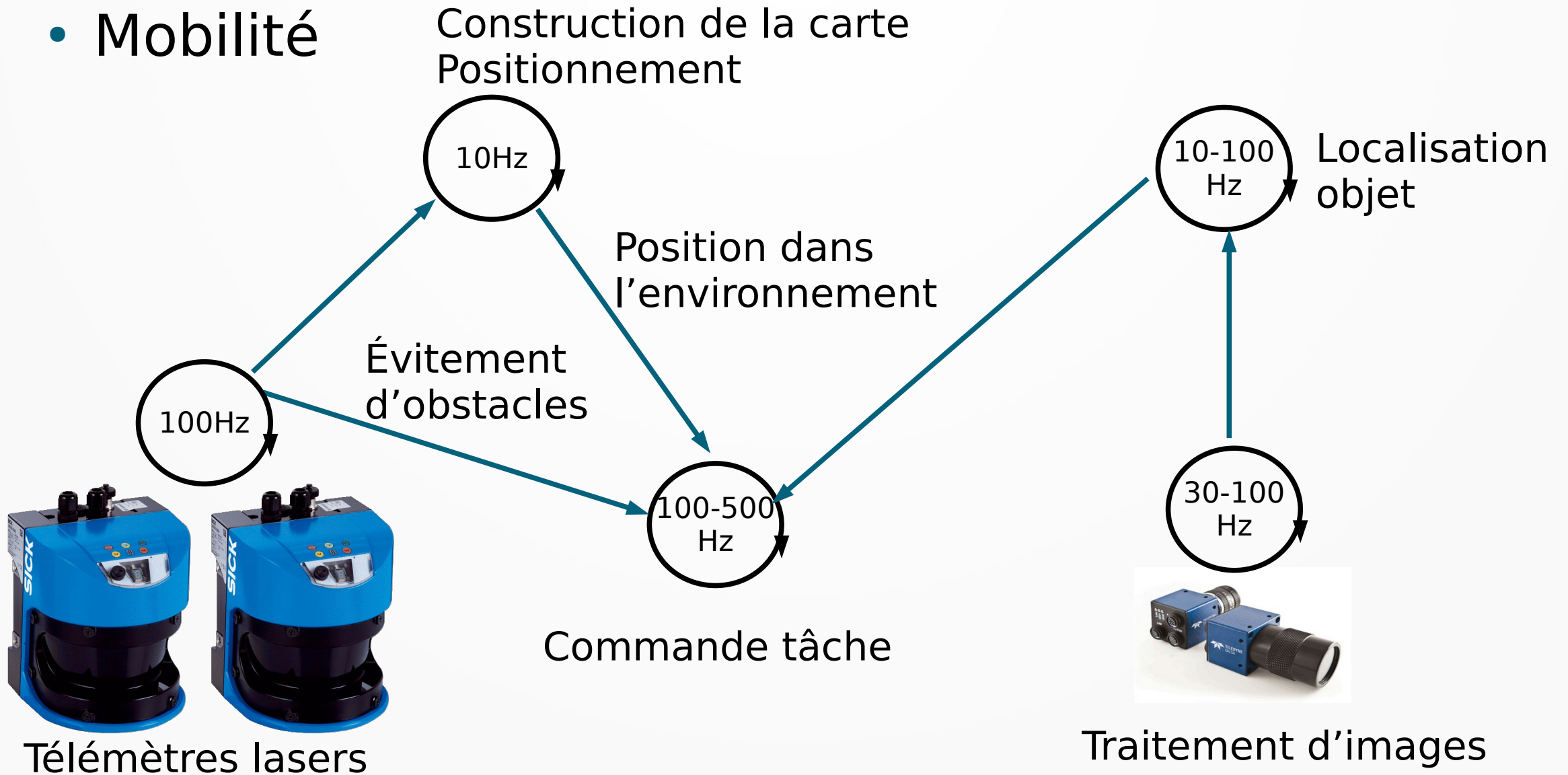


Architecture logicielle

- Localisation 3D de la personne

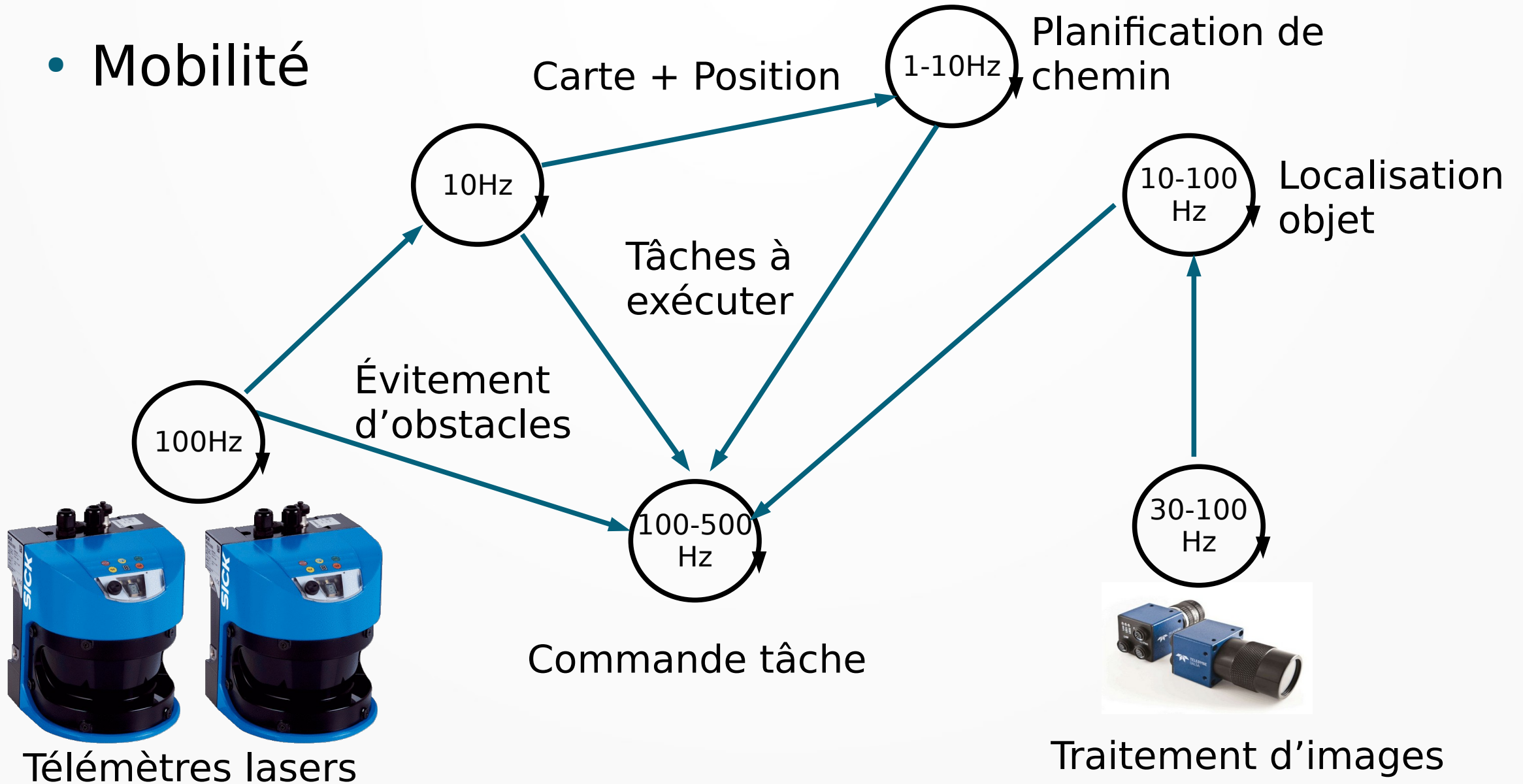
Architecture logicielle

- Mobilité



Architecture logicielle

- Mobilité



Perception de l'environnement

- Mobilité
 - SLAM : Localisation & Cartographie (e.g. algos ICP)
 - Évitement d'obstacles
 - Planification de chemin (e.g. algo A*)
 - Suivi de chemin

Architecture logicielle

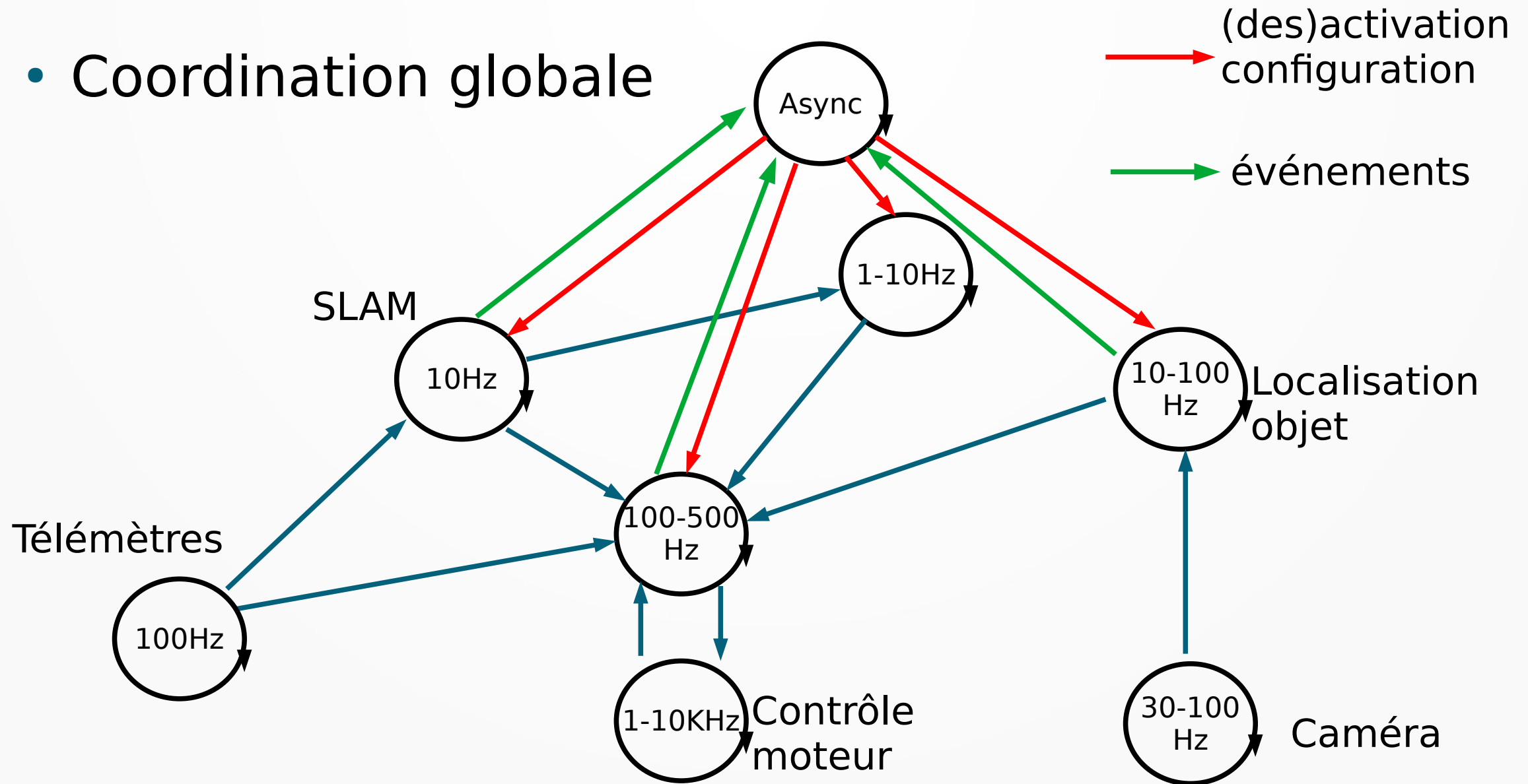
- SLAM

Architecture logicielle

- Anticipation et évitement d'obstacles

Architecture logicielle

- Coordination globale



Architecture logicielle

- Coordination globale
 - Modèle d'exécution asynchrone (FSM)
 - Configuration :
 - Définir les tâches à exécuter, les consignes associées aux tâches
 - Critères de détection
 - Événements
 - Objet/personne détectée
 - Tache terminée

Architecture logicielle

- État des lieux
 - Architecture logicielle des robots gagne en complexité
 - De nombreux types d'algorithmes
 - Variété de matériels
 - Des contraintes temps-réels à gérer
 - Des outils (ROS) existent pour gérer cette complexité
 - Facile d'utilisation
 - Pas de réflexion méthodologique poussée

Plan

- Introduction
- Principes de base
- Architecture logicielle
- **Nouvelles tendances**
- Conclusion

Programmation de mission

- Objectifs : comportements autonomes complexes
- Programmation « simplifiée » de plus haut niveau
 - Déclaration des fonctionnalités et ressources disponibles
 - Définition de comportement de plus haut niveau (behaviors, skills)
 - Composition de comportements
 - Mission : Comportement de plus haut niveau
 - Peut être décrit par un non expert
 - Peut être (partiellement) planifié

Programmation de mission

- Exemple d'outil : les behavior trees
 - Issu du jeu vidéo

Behavior Developer
Behavior Tree using Groot

Groot
The Fancy Behavior Tree Editor

System Builder
Intralogistics 4.0 Pilot using the SmartMDSD Toolchain

actions in behavior trees represent and call skills

Skill:
MoveBaseToGoal \$goal:
laserComponent.state = active
plannerComponent.goal = \$goal_coordinate

Component Supplier

Skills provide access to the functionalities realized within components and make them accessible to the task level. Skills coordinate software components through the RobMoSys Software Component Coordination Interface. With skill definitions on RobMoSys Tier 2, skills enable the task modeling independent of the underlying software component architecture. Skill implementations are bundled with software components and provided by the component supplier.

configure laser component

configure planner component

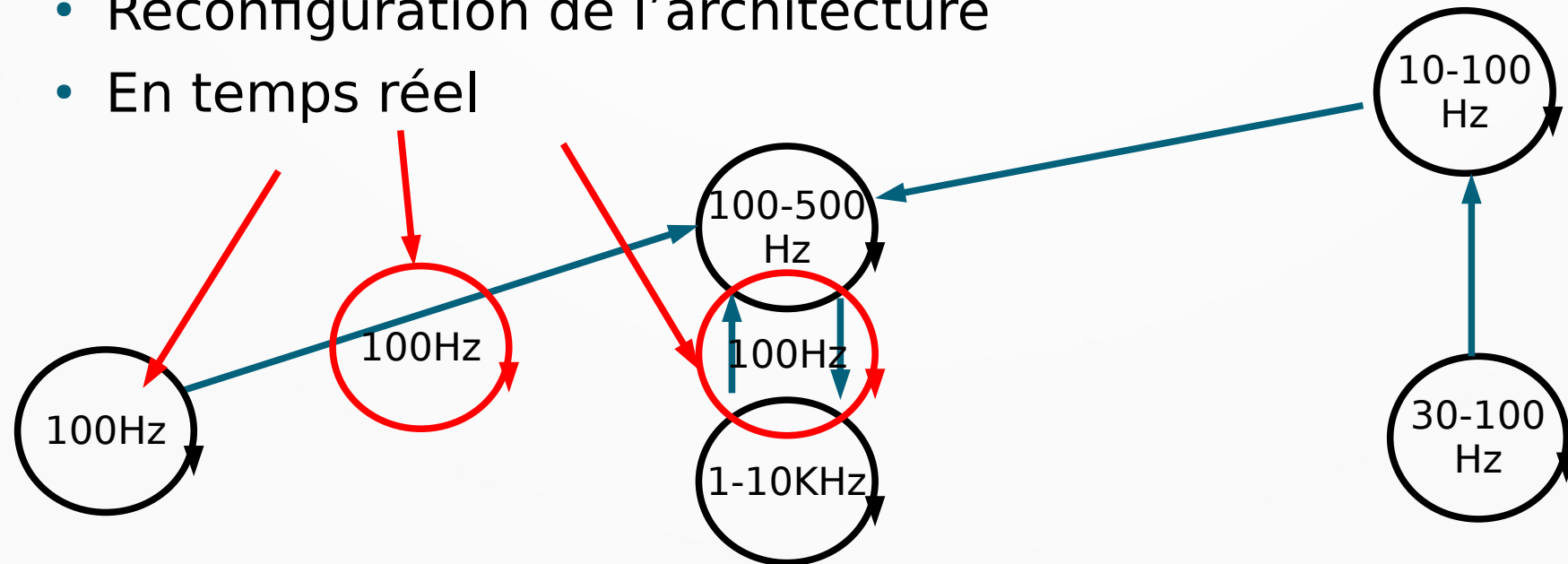
RobMoSys

Programmation de mission

- Exécution de missions/comportements
 - Assurer la tolérance aux fautes
 - Détection et Diagnostic de fautes
 - Recouvrement automatique
 - Garantir le respect de critères de performance
 - Permet de faire des choix automatique pour le recouvrement de fautes
 - Automatique et contextuel
 - Exemple : le robot a assez d'énergie pour revenir à sa base

Programmation de mission

- Détection et Diagnostic
 - Permanent : matériel/drivers/aspects système
 - Contextuel : enf fonction de l'état de la mission/environnement
 - Problème : ajouter/enlever du code de détection d'erreur
 - Reconfiguration de l'architecture
 - En temps réel



Programmation de mission

- Recouvrement et respect de critères de performance
 - Calculer une manière d'exécuter le plan/comportement
 - Respectant des critères de performance
 - Prenant en compte l'état du robot (pannes)
 - Si pas de solution : recouvrement = re-planification de la mission

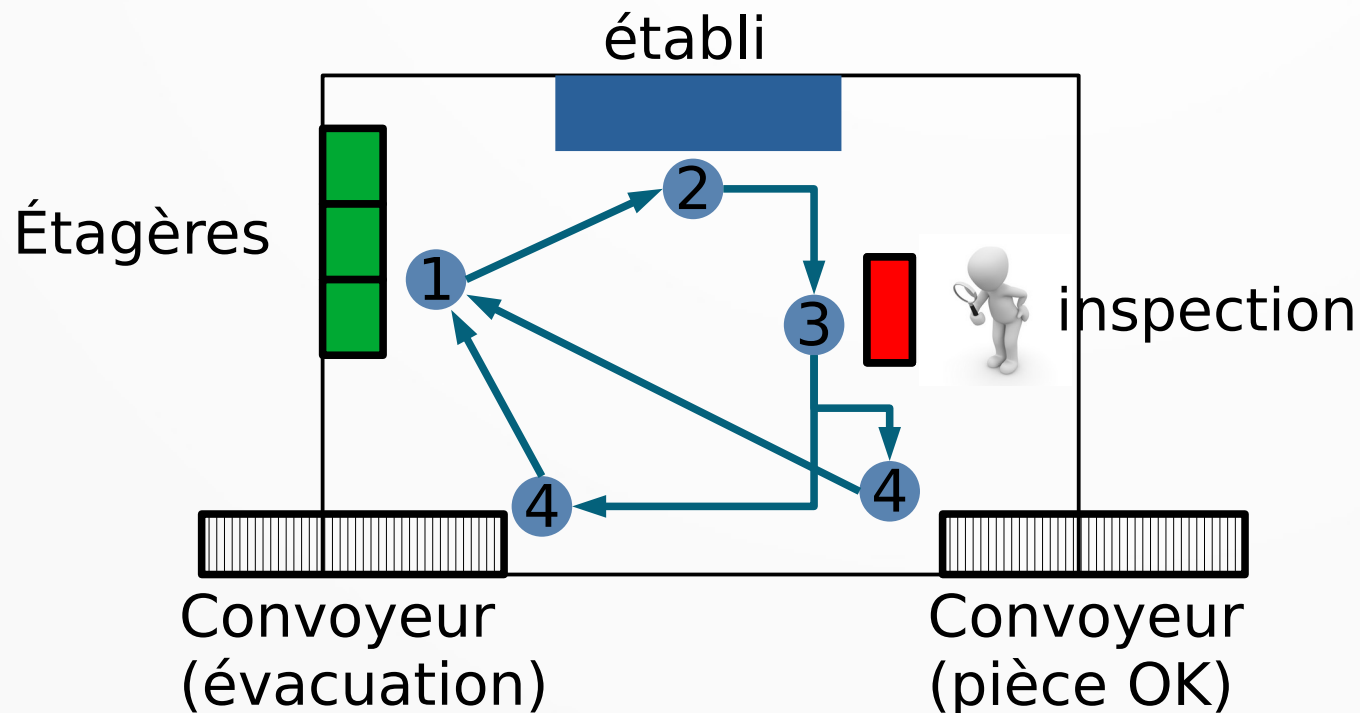
Programmation de mission

Programmation intuitive

- Principe : programmer le robot en utilisant ses capacités propres
 - Définir de nouveaux comportements/mission
 - téléopération
 - collaboration physique/visuelle/verbale
 - En utilisant :
 - ses comportements disponibles de base
 - Sans utiliser :
 - De langage de programmation...
 - Explicitement des descriptions de comportement

Programmation intuitive

- Scénario « idéal » : cellule d'assemblage
 - Apprendre au robot à assembler une pièce



- 1 Prendre les éléments
- 2 Assembler les éléments
- 3 Attendre décision humain
- 4 Évacuer l'objet

Programmation intuitive

- Exemple : utiliser la collaboration physique pour « montrer » les gestes à réaliser

Programmation intuitive

- Exemple : utiliser la collaboration :
 - Physique : pour définir les points les actions
 - Visuelle : pour donner des ordres directs

Programmation intuitive

- Problèmes ouverts :
 - Apprendre des gestes de manipulation complexe
 - Compréhension des actions à réaliser sur l'objet
 - Acquérir une connaissance minimale de l'objet (au moins sa géométrie)
 - Compréhension minimale de la scène
 - « Prendre tel type d'objet dans les étagères » et pas « Prendre tel objet à telle position »
 - Détection & positionnement des objets de la scène et leurs relations
 - Etc.

Programmation intuitive

- Utiliser des dispositifs de réalité virtuelle/augmentée
 - Visualiser ce que le robot apprend
 - nouveaux comportements
 - Définir des éléments virtuels
 - positions à atteindre dans l'espace
 - Zone d'exclusion
 - Généralement : définir via des interfaces ce que le robot ne sait pas apprendre
 - type d'objet à détecter, modèle de l'objet

Plan

- Introduction
- Principes de base
- Architecture logicielle
- Nouvelles tendances
- **Conclusion**

Conclusion

- Beaucoup de sujets « informatiques »
 - Ingénierie logicielle :
 - Amélioration des middleware (aspect temps réel)
 - Langages/modèles de composants et d'architecture
 - Langages/Modèles de programmation de comportements
 - Intelligence Artificielle :
 - Planification de mission/comportements
 - Apprentissage automatique (statistique ET symbolique)

Merci pour votre attention

Question ?