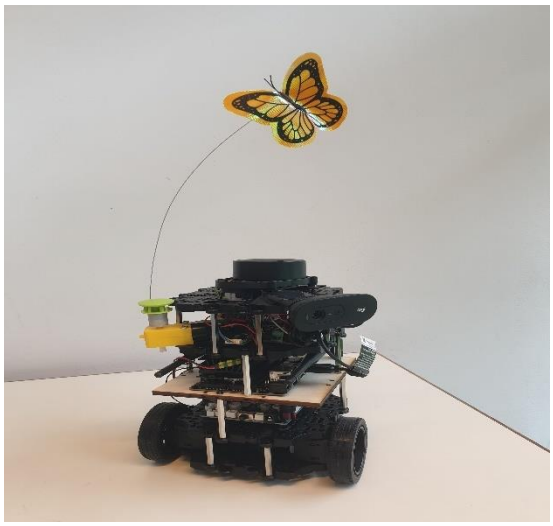
Rapport de Projet : CATBOT - Robot Compagnon pour Chats



Travail présenté **par Aimée BOUKAMBA & Agathe DELFLY** sous la direction de Monsieur **Frédéric JUAN**.

Sommaire :

Introduction

1.1 Contexte du Projet

Dans le cadre de notre projet innovant, nous présentons CATBOT, un robot compagnon intelligent spécialement conçu pour répondre aux besoins des chats domestiques. Basé sur la plateforme robotique TurtleBot 3, notre solution intègre des technologies avancées de vision par ordinateur et d'intelligence artificielle pour offrir un système autonome et interactif.

Notre robot combine plusieurs fonctionnalités clés :

- Une reconnaissance précise des chats via l'algorithme YOLO

- Des capacités d'interaction

- Une surveillance comportementale en temps réel

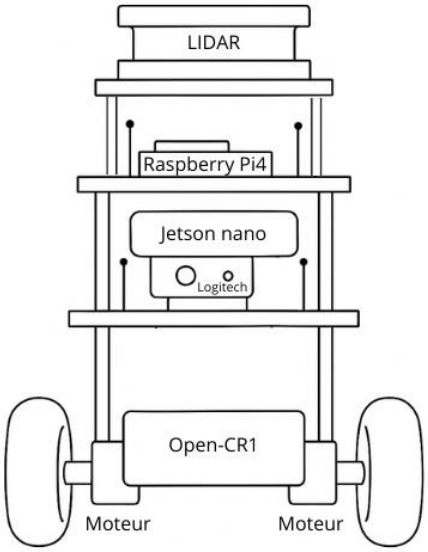
1.2 Objectifs

- **Autonomie** : Navigation ROS1 et détection d'obstacles.
- **Interaction Chat-Robot** : Détection du chat (YOLO), suivi et réponses adaptées.
- **Alimentation Automatisée** : Système de distribution de nourriture contrôlé à distance.
- **Surveillance** : Analyse du comportement du chat (machine learning).

Ce rapport détaillera notre approche technique, les défis rencontrés lors du développement, ainsi que les solutions innovantes que nous avons mises en œuvre. Nous présenterons également les résultats obtenus et les perspectives d'amélioration pour faire de CATBOT une solution toujours plus performante et utile au quotidien.

Architecture du Système

2.1 Matériel Utilisé



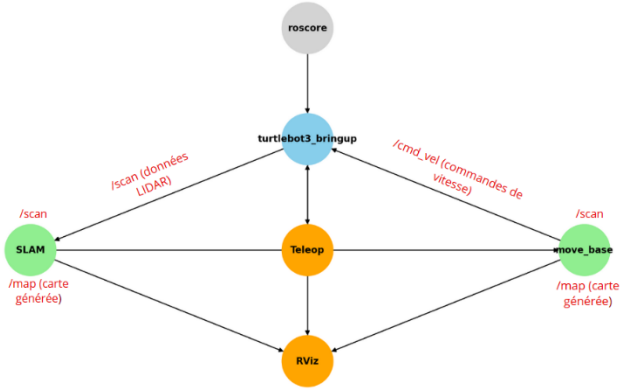
- **TurtleBot 3 Burger**
  - Base mobile avec Lidar (SLAM pour la navigation).
  - Raspberry Pi 4.
  - Jetson Nano + Caméra pour le traitement IA.
- **Module de Distribution de Nourriture (Amélioration future)**
- **Système de Communication**
  - MQTT / ROS1 pour les échanges entre Jetson et Raspberry Pi.

2.2 Logiciels et Outils

2.2.1 **ROS1 Noetic** (pour la navigation autonome).

ROS est un framework open-source dédié au développement d'applications robotiques. Il fournit une collection d'outils, de bibliothèques et de conventions pour simplifier la création de systèmes robotiques complexes. Dans le cadre du CATBOT, ROS nous permet de :

- Gérer la **communication** entre les différents composants matériels (capteurs, actionneurs)
- Coordonner les **traitements temps réel** (vision, contrôle moteur)
- Faciliter le **débogage** et la visualisation des données
- Bénéficier d'une large communauté et de packages pré-existants



Nous utilisons **ROS 1 noetic** principalement parce que le fournisseur nous le recommande mais nous envisageons de passer à **ROS 2 Humble** pour sa meilleure gestion des systèmes temps réel et ses améliorations en termes de sécurité et de performances réseau.

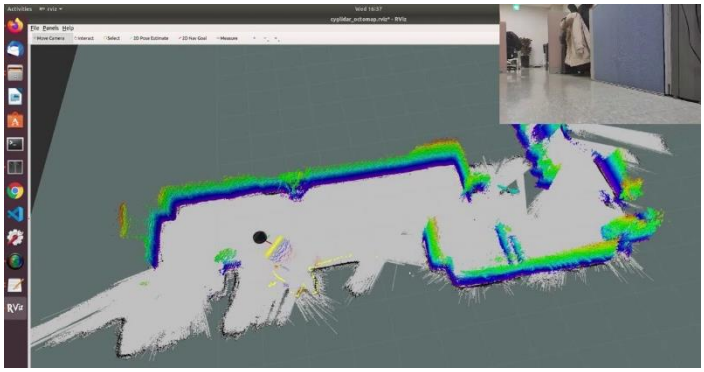
2.2.2 SLAM (Simultaneous Localization and Mapping)

Le SLAM permet au robot de :

1. Cartographier son environnement en temps réel à l'aide du LIDAR du TurtleBot 3 (mesures précises de distance)
2. Se localiser dans cette carte tout en se déplaçant

Le SLAM est essentiel pour que le CATBOT :

- Navigue sans collision
- Mémorise les zones importantes (gamelle, litière)
- Optimise ses trajets



2.2.3 Téléopération

Le système de téléopération permet :

- Un **contrôle à distance** via :
- La **supervision** en temps réel :

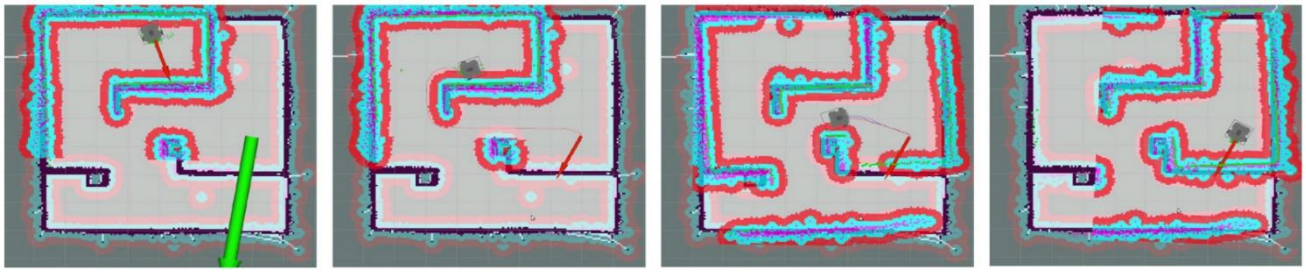
Cette fonctionnalité utilise le package **teleop_twist_keyboard** adapté pour :

- Recevoir des commandes via WiFi
- Gérer les priorités entre autonomie et contrôle manuel
- Assurer une latence minimale (< 100ms)

2.2.4 Navigation Autonome

Notre système de navigation repose sur **MoveBase** (ROS) pour :

- La planification de trajectoire
- L'évitement d'obstacles



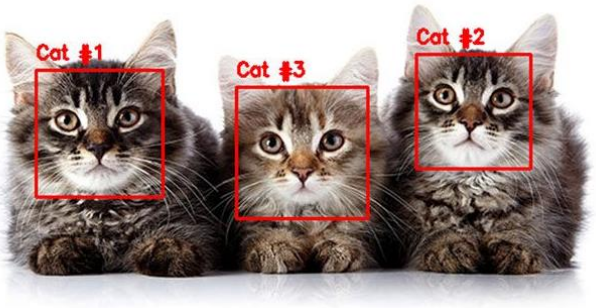
2.2.5 Interaction jouet

YOLO est un algorithme de détection d'objets en temps réel qui traite les images en une seule passe réseau, offrant un excellent compromis entre vitesse et précision. Contrairement aux méthodes traditionnelles, YOLO :

- Analyse l'image globale en une seule évaluation
- Atteint des performances en temps réel (> 30 FPS sur GPU)
- Est particulièrement adapté pour la détection d'animaux domestiques

Nous utilisons **YOLOv8**, la dernière version, pour ses améliorations en :

- Précision de détection des petits objets
- Efficacité sur matériel embarqué
- Facilité d'entraînement sur des datasets personnalisés



Initialement, nous avons tenté d'exécuter YOLOv8 sur un **Raspberry Pi 4** (4GB) :

Problèmes rencontrés :

1. **Latence excessive** (>5s par image)
2. **Surchauffe** du processeur
3. **Précision insuffisante** (taux de détection <60%)

Causes identifiées :

- Puissance de calcul insuffisante pour l'inférence YOLO
- Mémoire RAM limitée pour les modèles non optimisés
- Absence d'accélération matérielle GPU

Nous avons alors opté pour une migration vers une **Jetson Nano** offrant :

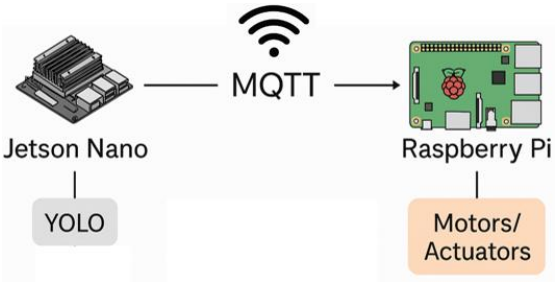
- Un GPU NVIDIA dédié
- 128 coeurs CUDA
- Support natif des frameworks d'IA

Cette intégration nous a forcé à trouver une méthode de communication entre la jeston et le raspberry pi **: MQTT.**

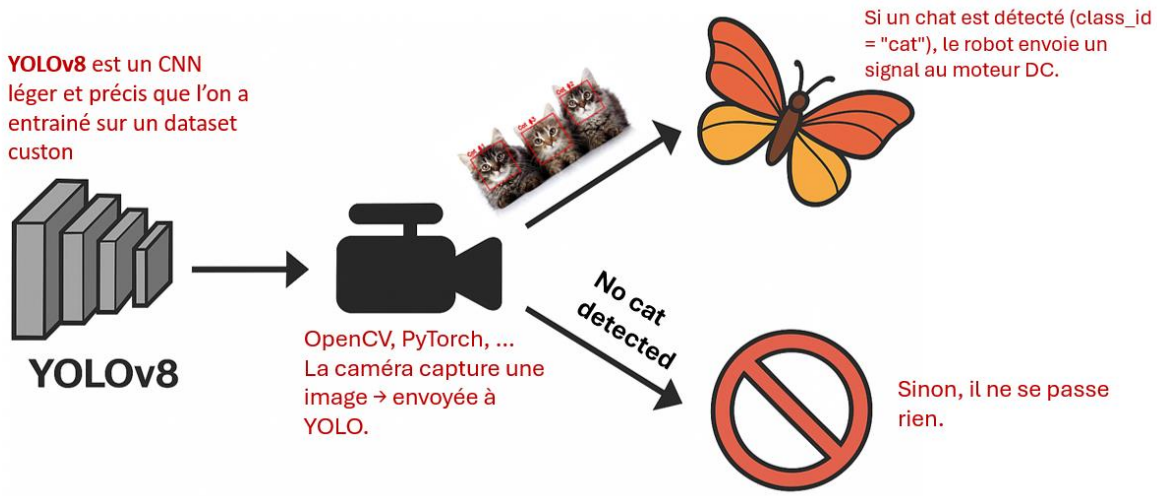
Pour la communication entre la Jetson (vision) et le Raspberry Pi (contrôle) :

Architecture MQTT :

- **Broker** : Mosquitto sur le Raspberry Pi
- **Topics** :
  - catbot/detection (messages JSON)
  - catbot/commands (contrôle du jouet)



Pour permettre au Cabot d'interagir avec un chat. Nous avons intégré un jouet rotatif au robot. Pour cela, nous avons ajouté un driver L298 et un moteur à courant continu alimenté en 5 V par le Raspberry pi. Lorsqu'un chat est détecté avec YOLO, le jouet est activé et tourne pendant 2 minutes afin de divertir le chat.



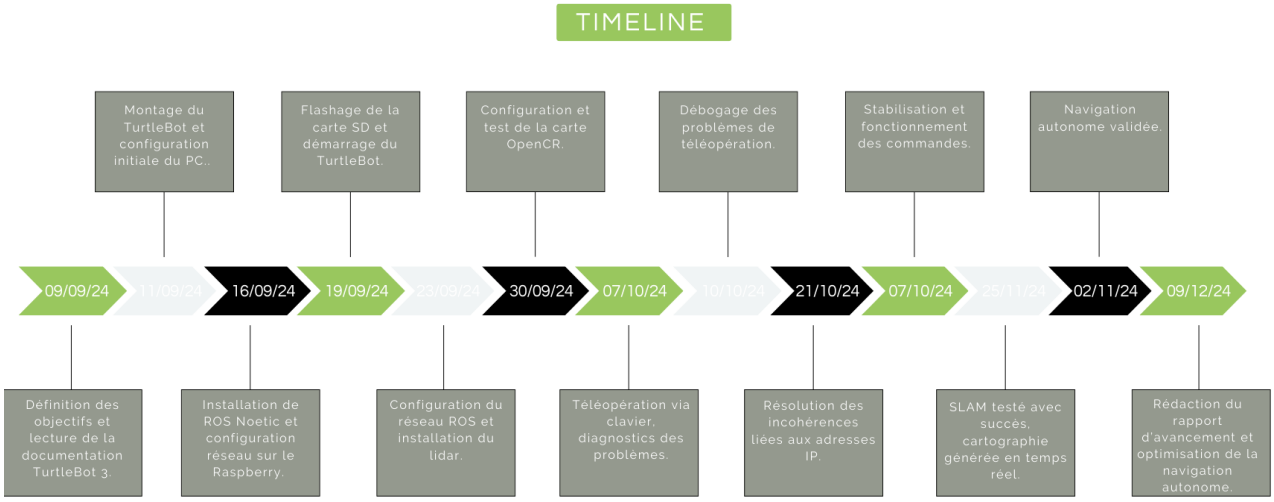
3. Détail des Coûts Réels du Projet

Composant	Description	Prix (€)	Remarques
TurtleBot 3 Burger	Inclut Raspberry Pi 4 et caméra de base	500	Pas besoin d'acheter RPi séparément
Jetson Nano Developer Kit	Modèle 4GB	175	Essentiel pour YOLOv8
Caméra Logitech Standard	Modèle C270 (non Pro)	35	Suffisante pour la détection
Module L298N	Contrôleur moteur	8	Pour le système de distribution
Moteur CC + Accessoires	Moteur jaune 6V + pièces 3D	15	Coût total jouet/modifications
Divers Électroniques	Câbles, connecteurs, alimentation	25	Frais incompressibles

Total Réel : 758 €

Budget optimisé grâce à une meilleure allocation des composants

Le projet s'est déroulé en deux phases principales. La première phase a été consacrée à la prise en main du Turtlebot et à la navigation autonome.



Cependant que la seconde phase a impliqué une répartition des tâches pour finaliser le projet sans planning précis.

4. Améliorations futures et perspectives

➢ Optimisation Matérielle :

Centralisation sur Jetson Nano

- Remplacer le Raspberry Pi (inclus dans le TurtleBot) par une **Jetson Nano unique** pour :
  - Unifier le traitement vision (YOLOv8) et la logique ROS
  - Bénéficier du **GPU intégré** pour les calculs IA
  - Simplifier l'alimentation (une seule carte à alimenter)
- **Solution mémoire** :
  - Ajout d'un **SSD NVMe** via port M.2 (pour stocker les logs de détection et datasets)

Système de distribution de nourriture :

- Compartiment étanche imprimé en 3D avec :
  - **Trémie réglable** pour contrôler la quantité de croquettes
  - **Moteur pas-à-pas** (au lieu du moteur CC) pour une distribution précise
  - **Capteur de niveau** (ultrason ou IR) pour alerter en cas de faible stock

Coque de protection :

- Design ergonomique en PLA/PETG pour :
  - Protéger les câbles et circuits
  - Améliorer l'esthétique (forme arrondie, adaptée aux animaux) & intégrer des **LEDs interactives** (feedback visuel pour le chat)

Caméra haute performance :

- Remplacement de la Logitech C270 par une **caméra Intel RealSense D435** :
  - Profondeur (RGB-D) pour une meilleure détection 3D du chat
  - Résolution 1080p à 60 FPS (idéal pour YOLO en mouvement)

- **Migration vers ROS 2**

Avantages Clés

- **Meilleure gestion des ressources** :
  - Découplage des nodes (ex : séparer la détection YOLO du contrôle moteur)
  - Latence réduite grâce au **DDS** (protocole de communication)
- **Support long-terme** :
  - ROS 1 (Noetic) arrive en fin de vie → migration obligatoire
- **Fonctionnalités avancées** :
  - Sécurité intégrée (cryptage des topics MQTT)
  - Meilleure gestion des temps réels (pour les mouvements du robot)

Tâches pour le Stage ROS

1. **Adapter les packages existants** :
   - Refaire les nodes en Python 3 (ROS 2 utilise uniquement Python 3)
   - Mettre à jour les dépendances (ex : cv_bridge pour OpenCV 4)
2. **Recalibrer la navigation** :
   - Recartographier l'environnement avec **Nav2** (SLAM ROS 2)
   - Optimiser les paramètres de move_base pour les espaces domestiques
3. **Tests de robustesse** :
   - Vérifier la stabilité sur des sessions prolongées (24h+)

Conclusion

Ce projet nous a permis de mettre en pratique nos connaissances en robotique et vision par ordinateur à travers un système concret. Forts de cette expérience, nous poursuivrons le développement l'année prochaine en nous appuyant sur nos stages respectifs :

- **Stage en vision par ordinateur** pour optimiser la détection des chats (YOLOv8 sur Jetson) et intégrer une caméra plus performante
- **Stage en ROS** pour migrer vers ROS 2 et améliorer la navigation autonome

L'objectif est d'aboutir à un prototype complet avec distribution automatique de nourriture et une meilleure interaction chat-robot.