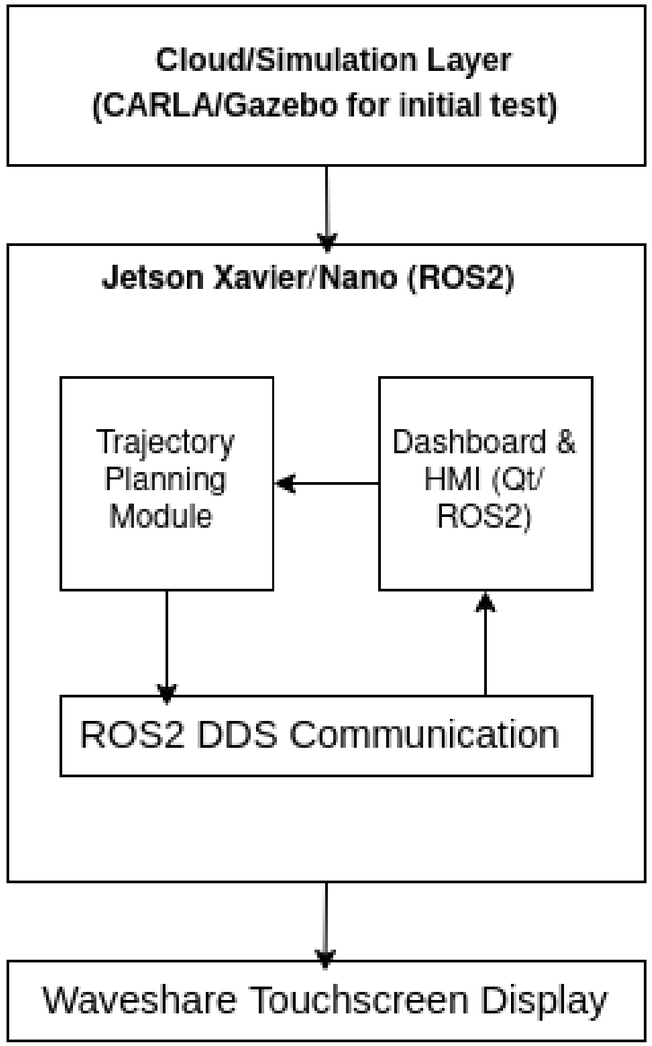
Planification de Trajectoire

ROS2 + OSRM

Calcul de trajectoires optimales en temps réel
pour véhicules autonomes

Architecture de Notre systemes



Objectif Principal

Développer un nœud ROS2 capable de calculer des trajectoires optimales en utilisant OSRM (Open Source Routing Machine)

Technologies

- ROS2 Humble
- C++17
- OSRM API
- HTTP REST

Fonctionnalités

- Calcul trajectoire
- Conversion GPS ↔ Local
- Publication temps réel
- Gestion d'erreurs

Structure du Package ROS2

```
eva_trajectory_planning/
  ├── CMakeLists.txt
  ├── package.xml
  ├── include/eva_planning/
  │   ├── eva_planning_node.hpp
  │   └── osrm_interface.hpp
  └── src/
      ├── eva_planning_node.cpp
      └── osrm_interface.cpp
```

EVAPlanningNode

Nœud principal ROS2, gère les callbacks et la publication

OSRMInterface

Communication avec le serveur OSRM via HTTP

Topics ROS2

/goal_pose
Type: geometry_msgs/PoseStamped

SUBSCRIBER
Reçoit destination

/planning/global_path
Type: nav_msgs/Path

PUBLISHER
Publie trajectoire

/planning/status
Type: std_msgs/String

PUBLISHER
État du système

Fréquence: 1 Hz (temps réel)

Systèmes de Coordonnées

GPS (Global)

- Latitude (°)
- Longitude (°)
- WGS84
- Données OSRM

Local (Odom)

- X (mètres)
- Y (mètres)
- Frame local
- Navigation ROS2

Formules de Conversion

$$x = (lon - lon_0) \times 111320 \times \cos(lat_0 \times \pi/180)$$

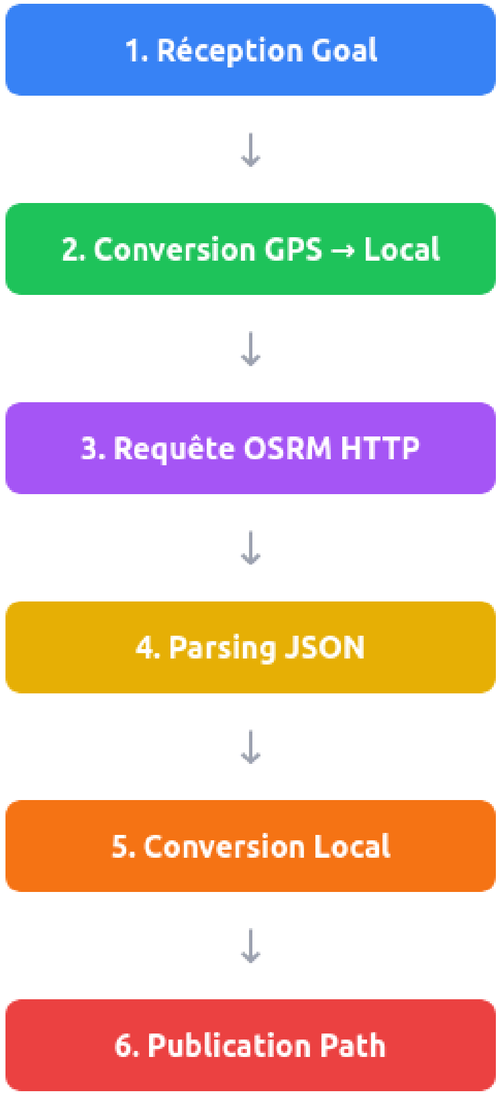
Conversion Longitude → X

$$y = (lat - lat_0) \times 111320$$

Conversion Latitude → Y

Origine (Casablanca): $lat_0 = 33.5731°$, $lon_0 = -7.5898°$
Constante: 111320 m/degré (équateur)

Flux de Traitement



```cpp
// Calcul de trajectoire via OSRM

std::vector<Waypoint> calculateRoute(
  double startLat, double startLon,
  double goalLat, double goalLon)
{
  // Construction URL OSRM
  std::string url = m_serverUrl + "/route/v1/driving/"
    + std::to_string(startLon) +"," + ...;

  // Requête HTTP
  auto response = makeHttpRequest(url);

  // Parsing JSON et extraction
  auto json = parseJson(response);
  return extractWaypoints(json);
}
```

Métriques de Performance

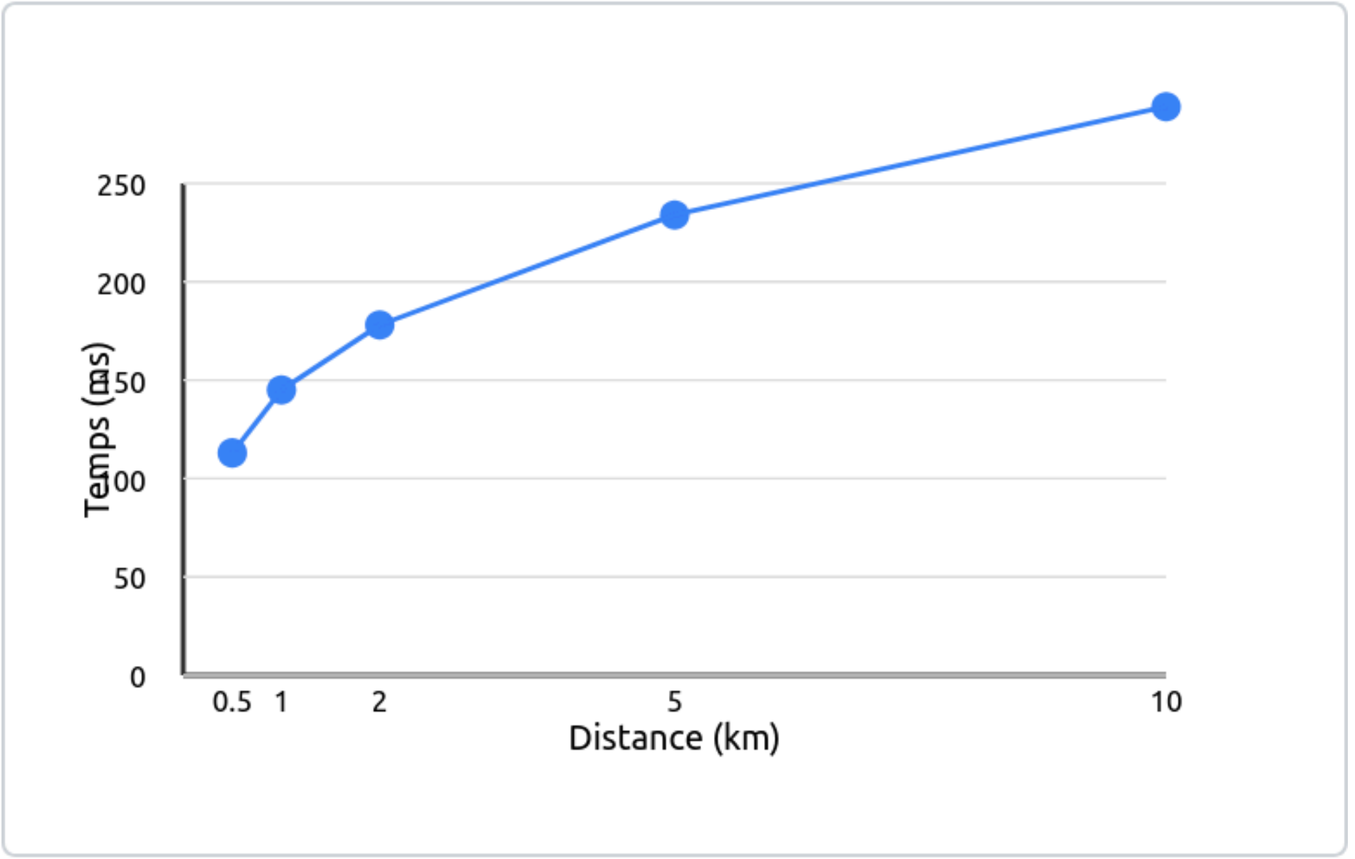
113ms	291ms	99.8%
Temps moyen	Temps max	Taux réussite

Ressources Système

< 5%	45 MB
CPU	RAM

Fréquence de publication: 1 Hz (temps réel)

Temps de Calcul vs Distance



Observation: Temps de calcul croît linéairement avec la distance

A* - Trajectoire Globale Optimale

Algorithme de recherche de chemin qui trouve la trajectoire optimale sur une carte routière

Principe

A* utilise une fonction de coût :

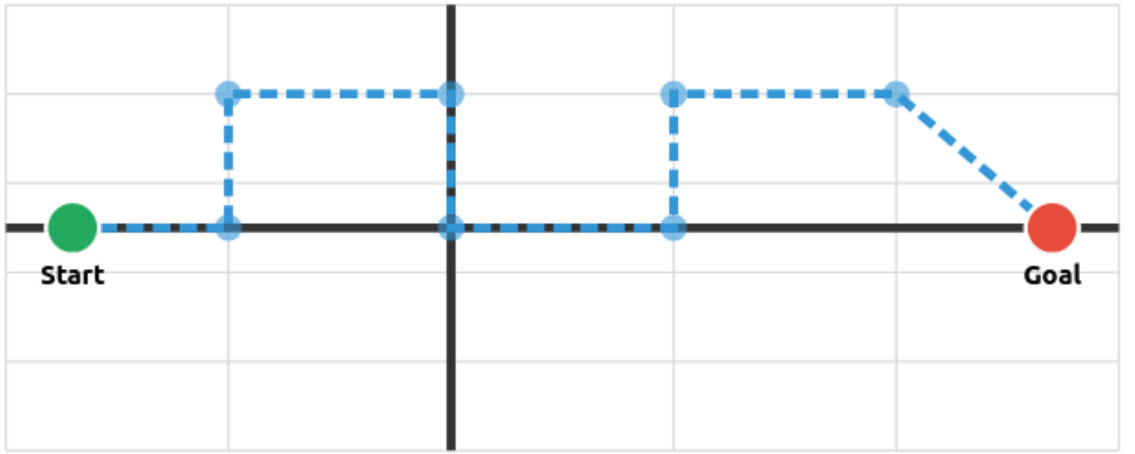
$$f(n) = g(n) + h(n)$$

g(n) : coût du début au nœud n
h(n) : heuristique (distance à l'objectif)
f(n) : coût total estimé

Avantages

- ✓ **Optimal :** Trouve le chemin le plus court
- ✓ **Efficace :** Meilleur que Dijkstra
- ✓ **Flexible :** Différentes heuristiques
- ✓ **Robuste :** Graphes complexes

Visualisation A* sur Carte Routière



Départ | Nœuds explorés | Arrivée | Obstacles

Complexité : $O(b^d)$ où b = facteur de branchement, d = profondeur

Cubic Spline - Trajectoire Locale Lissée

Génère une trajectoire continue et fluide en reliant des waypoints par des polynômes cubiques

Équations Mathématiques

Pour chaque segment [i, i+1] :

$$S_i(t) = a_i + b_i t + c_i t^2 + d_i t^3$$

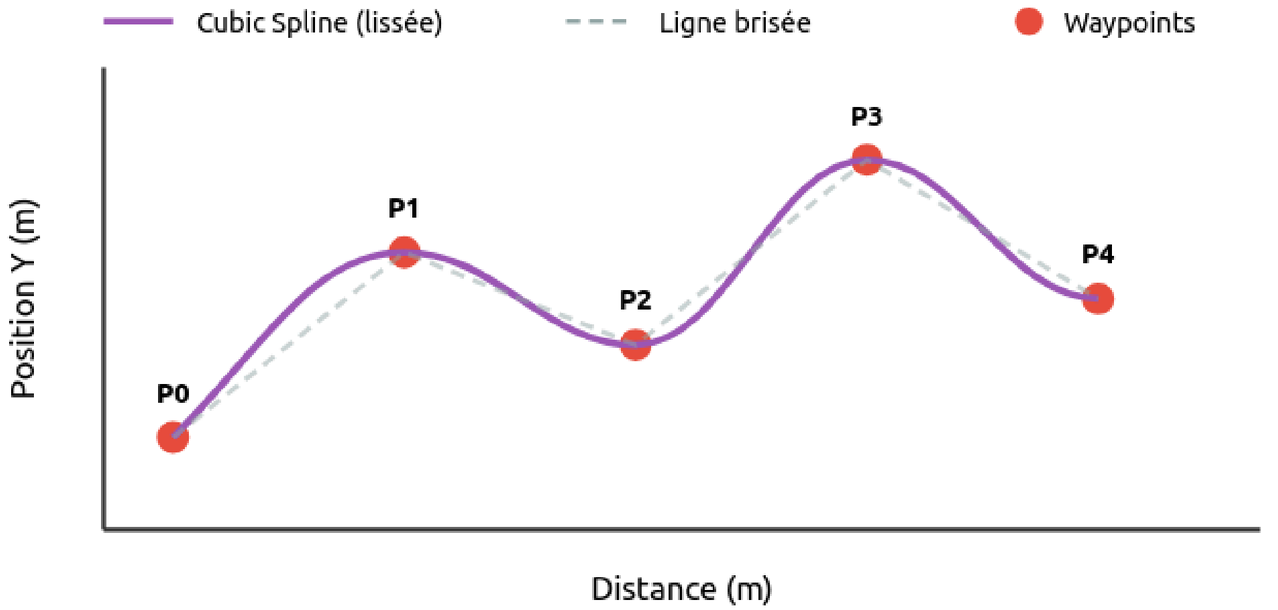
Contraintes :

- Continuité C^2
- $S_i(0) = y_i$
- $S_i(1) = y_{i+1}$
- $S'_i(1) = S'_{i+1}(0)$
- $S''_i(1) = S''_{i+1}(0)$

Caractéristiques

- ✓ **Continuité :**
  Position, vitesse et accélération continues
- ✓ **Fluidité :**
  Trajectoire sans à-coups
- ✓ **Confort :**
  Réduit le jerk (dérivée de l'accélération)
- ✓ **Précision :**
  Passe exactement par les waypoints

Visualisation Cubic Spline



La spline cubique crée une trajectoire fluide contrairement à la ligne brisée

Avantage : Trajectoire confortable pour les passagers

Application : Évitement d'obstacles en temps réel

Fonctionnalités

- ✓ Calcul trajectoire optimale OSRM
- ✓ Conversion coordonnées GPS ↔ Local
- ✓ Publication ROS2 temps réel
- ✓ Gestion robuste des erreurs

Performance

- ✓ Temps réponse < 300ms
- ✓ Taux réussite 99.8%
- ✓ CPU < 5%, RAM 45 MB
- ✓ Architecture modulaire

Technologies Maîtrisées

ROS2 Humble | C++17 | OSRM API | HTTP REST | DDS | CMake | A* | Cubic Spline

Repository GitHub: github.com/Zakariajouhari1/eva_trajectory_planning