

Master Systèmes Dynamiques et Signaux

Mémoire

Génération de trajectoires et commande de robots mobiles pour l'évitement d'obstacles dynamiques

Auteur :
M. Harendra RANGARADJOU

Jury :
Pr. L. Hardouin
Pr. L. Jaulin
MdC.-HDR L. LAPIERRE
MdC. F. LE BARS
MdC. S. ROHOU

Version du
25 août 2025

Remerciements

Je tiens à remercier mon tuteur **Martín MUJICA (MCf.)**, **Rémi PORÉ (doctorant)** et **Viviane CADENAT (MCf.-HDR)** pour leur encadrement et les précieux conseils qu'ils m'ont apportés tout au long de ce projet de fin d'études. Je remercie tout particulièrement Viviane CADENAT et Martín MUJICA pour le soutien qu'ils ont accordé à ma candidature au sujet de thèse que j'aurai la chance de débuter à la rentrée prochaine.

Je tiens enfin à exprimer ma gratitude envers l'ensemble de l'équipe enseignante de la spécialité Robotique Autonome de l'*ENSTA* Bretagne pour la qualité de l'enseignement dispensé durant ma formation, malgré des conditions parfois difficiles.

Résumé

Dans le cadre de ma dernière année de formation d'ingénieur à l'*ENSTA* Bretagne, j'ai eu l'opportunité d'effectuer un stage de cinq mois et demi au sein de l'équipe Robotique, Action, Perception (*RAP*) du Laboratoire d'Analyse et d'Architecture des Systèmes (*LAAS*), à Toulouse. Le stage, devant initialement porter sur l'adaptation du travail réalisé sur un manipulateur mobile d'extérieur au cas d'un manipulateur mobile d'intérieur, a rapidement basculé sur le développement d'une solution distincte de celle déjà en place. J'ai ainsi pu réaliser de nombreuses tâches ayant trait à la coordination des déplacements d'un manipulateur mobile dans un environnement dynamique. J'ai notamment travaillé sur l'implémentation d'algorithmes de planification de trajectoires, de contrôle et de guidage pour la base mobile du robot.

Ce rapport vise dans un premier temps à établir le contexte du projet. Le sujet est présenté et une analyse de l'état de l'art est réalisée afin de justifier la problématique et la méthodologie de travail retenues. Les objectifs ainsi que la planification et le déroulement des activités sont également traités. Dans un deuxième temps, les travaux effectués sont détaillés. Les algorithmes développés et leurs implémentations sont présentés. Les expérimentations en simulation et en conditions réelles sont décrites. Dans un troisième et dernier temps, les résultats obtenus sont exposés, suivis des difficultés notables rencontrées ainsi que les améliorations possibles.

Abstract

As part of my final year of training at *ENSTA* Bretagne, I undertook an internship within the Robotics Action Perception (*RAP*) team at the Laboratory for Analysis and Architecture of Systems (*LAAS*), in Toulouse. This internship, which initially aimed at adapting the work done on an outdoor mobile manipulator to an indoor one, quickly shifted towards the development of a distinct solution from the one already in place. As a result, I have completed various tasks related to the coordination of a mobile manipulator's movements in a dynamic environment. I have worked on the implementation of path planning, control, and guidance algorithms for the mobile base of the robot.

This report first aims to establish the context of the project. The topic is introduced and a review of the literature is carried out in order to justify the problem statement and the chosen methodology. The goals, as well as the planning and execution of activities, are also addressed. Secondly, the work done is described in detail. The developed algorithms and their implementations are presented. Experiments conducted both in simulation and in real-world conditions are outlined. Finally, the results obtained are presented, followed by the main challenges encountered and the possible improvements.

Mots-clés

Génération de trajectoires, Commande prédictive non-linéaire (NMPC), Évitement d'obstacles, Environnement dynamique, ROS2, Gazebo

Table des matières

1 Introduction

1.1 Parcours et motivation

La spécialisation robotique de la formation d'ingénieur généraliste de l'ENSTA Bretagne est fondamentalement pluridisciplinaire et couvre un large éventail de domaines d'application de la robotique mobile autonome. Elle propose un équilibre idéal entre enseignements théoriques et pratiques, combinant cours magistraux classiques et projets d'application variés. La spécialité étant cependant axée sur la robotique mobile, avec un accent sur la robotique marine, les robots manipulateurs n'ont que très peu été abordés en cours. Ayant néanmoins eu l'occasion de travailler avec plusieurs manipulateurs mobiles lors de divers projets, comme les multiples participations à la coupe de France de robotique du club de robotique de l'école, je me suis découvert un certain intérêt pour ce type de robot. Motivé par ma curiosité, j'ai donc choisi un stage de fin d'études dans le domaine de la manipulation mobile.

J'ai également choisi de faire mon stage en laboratoire plutôt qu'en entreprise afin de me rapprocher du monde de la recherche. J'avais déjà entamé ce rapprochement avec mon stage de deuxième année à l'Université Aston et mon choix de poursuivre le master recherche Ingénierie des Systèmes complexes, Parcours Systèmes Dynamiques et Signaux, en parallèle de ma dernière année à l'ENSTA. Cette orientation vers la recherche s'explique par mon ambition de poursuivre mes études par un doctorat, avant de me diriger vers la recherche et le développement. Par ailleurs, cet effort a porté ses fruits avec mon admission pour une thèse sur la « Coordination des mouvements d'un manipulateur mobile par asservissement visuel prédictif multi-caméra pour la réalisation de tâches complexes dans un environnement encombré » à partir du 1^{er} octobre, sous la co-direction de Viviane CADENAT [1] et de Martín MUJICA [2].

1.2 Environnement de travail

Le LAAS-CNRS (Laboratoire d'Analyse et d'Architecture des Systèmes) est un laboratoire français situé à Toulouse, rattaché au Centre National de la Recherche Scientifique (*CNRS*). Il est reconnu à l'internationale pour son excellence scientifique et son implication dans des projets de recherche partenariale. Il fut fondé en 1968 par Jean Lagasse et George Giralt sous le nom de « Laboratoire d'Automatique et de ses Applications Spatiales », un an à peine avant les premiers pas de l'Homme sur la Lune. À l'aube des années 1970, les programmes « d'applications spatiales » avec le Centre National d'Études Spatiales (*CNES*) se faisant plus rares, il est décidé toutefois d'orienter davantage les recherches du laboratoire vers l'étude de l'automatique et des systèmes complexes. Le laboratoire est ainsi rebaptisé « Laboratoire d'automatique et d'analyse des systèmes » en 1973, avant de devenir plus tardivement le « Laboratoire d'Analyse et d'Architecture des Systèmes ».

À date du 1^{er} janvier 2025, la structure emploie 193 enseignants-chercheurs, 241 doctorants et post-doctorants, 93 *ITA* et *BIATSS*, 69 *ITA*/Chercheurs en CDD. L'organigramme est donné dans l'Annexe B. Le laboratoire est composé de 24 équipes réparties entre 6 départements scientifiques, à savoir : Décision et Optimisation (DO), Gestion de l'énergie (GE), Systèmes hyperfréquences et photoniques (HOPES), Micro Nano Bio Technologies (MNBT), Réseaux, Informatique, Systèmes de confiance (RISC) et Robotique (ROB). La recherche y est organisée autour des axes thématiques transverses de l'énergie, de l'espace, de l'industrie du futur, de la santé et de l'environnement ainsi que du transport et de la mobilité.

1. enseignante-chercheuse *HDR* au *LAAS*
2. enseignant-chercheur au *LAAS*

Mon stage s'est déroulé sous la tutelle de Martín MUJICA, au sein de l'équipe Robotique, Action, Perception (*RAP*), l'une des 3 équipes du département ROB avec les équipes Robotique et Interactions (*RIS*) et *GEPETTO*. L'équipe *GEPETTO* est centrée sur l'analyse et la génération de mouvement des systèmes anthropomorphes. C'est l'une des équipes phares en robotique humanoïde, unanimement reconnue pour son expertise en génération de mouvement. L'équipe *RIS* développe quant à elle un projet de recherche portant essentiellement sur les machines autonomes intégrant des capacités de perception, de raisonnement, d'apprentissage, d'action et de réaction. Enfin, les recherches de l'équipe *RAP* concernent la conception, le prototypage, l'implémentation et l'évaluation d'algorithmes pour la perception visuelle, la commande et la navigation référencées capteur, la manipulation mobile interactive, la modélisation et la fusion de données multi-capteur.

Mon sujet s'est inscrit dans la continuité des travaux de l'équipe. Il était lié au projet de thèse en cours de Rémi PORÉ [1] intitulée « Commande par reconnaissance gestuelle et contrôle de manipulateur mobile en environnement extérieur ». Traitant initialement de l'adaptation à un milieu intérieur de certains résultats obtenus par PORÉ, il a assez vite évolué pour se concentrer sur la recherche et l'implémentation de nouvelles méthodes de génération de trajectoires et d'évitement d'obstacles.

1.3 Problématique et travaux connexes

En effet, le sujet de mon stage a été modifié peu de temps après mon arrivée, avant que j'aie le temps de me pencher réellement sur le travail réalisé par PORÉ. Le problème est alors devenu de savoir si l'aspect prédictif du Model Predictive Control (*MPC*) pouvait être utilisé dans le cadre de la génération de trajectoires en ligne. Concrètement, la question principale qui est ressortie était de savoir s'il était possible d'appliquer cette génération de trajectoires par *MPC* dans le contexte de l'évitement d'obstacles dynamiques.

Il est à noter, qu'un doute a subsisté pendant les premiers mois du projet quant au système spécifique pour lequel les trajectoires devraient être générées. Bien qu'il ait été décidé plus tard qu'il était plus intéressant de se concentrer sur la base mobile, l'état de l'art a été réalisé en supposant que les trajectoires pourraient être générées aussi bien pour un manipulateur mobile dans son ensemble que pour sa base mobile seule.

Dans un contrôleur à plusieurs niveaux, un *reference governor* permet de générer des points fixes pour des contrôleurs de plus bas niveau, comme un PID, afin de garantir un meilleur asservissement. Historiquement, les premières implémentations de contrôleurs de ce type remontent aux années 1980, dans le contexte de l'asservissement de procédés industriels. Les points fixes étaient alors calculés en résolvant un problème d'optimisation à l'aide d'outils comme le Dynamic Matrix Control (*DMC*), comme expliqué dans [1]. C'est exactement ce que nous cherchons à faire ici, générer des trajectoires de référence pour un contrôleur de plus bas niveau, en résolvant un problème d'optimisation.

1. doctorant au *LAAS*

L'appellation *reference governor* semble cependant propre au domaine des procédés industriels, où les échelles de temps en jeu sont généralement incompatibles avec la commande en temps réel de robots mobiles. Fort heureusement, avec l'augmentation de la puissance de calcul des processeurs, de nombreuses architectures de contrôle à plusieurs niveaux ont été développées pour des applications de robotique mobile au cours des deux dernières décennies. Ce type d'architecture est particulièrement commune dans le domaine des véhicules autonomes, très prolifique en termes de recherche et développement ces dernières années. Toutefois, les trajectoires calculées à l'étape de la planification ne sont pas toujours obtenues par la résolution d'un problème d'optimisation dans la litérature. Des méthodes de planification de chemin plus traditionnelles sont en effet couramment utilisées, comme illustré par l'utilisation de champs de potentiel artificiels dans [2] ou [3]. De plus, quand le formalisme de la commande optimale est bien utilisé, les trajectoires optimales sont souvent déterminées en termes de commandes pour le véhicule, comme dans [4]. Ceci ne correspond pas à notre objectif de génération de trajectoires en termes d'états.

Quelques travaux utilisent le *MPC*, linéaire ou non, dans le cadre de la génération de trajectoires au sens qui nous intéresse, notamment [5] et [6] . Là où [5] traite de planification de trajectoires hors ligne, l'approche présentée dans [6] correspond plus à notre objectif d'évitement d'obstacles dynamique. De fait, cette dernière utilise un *MPC* afin d'obtenir des valeurs de références en temps réel pour un contrôleur de plus bas niveau, les valeurs en question étant les degrés de libertés des articulations d'un robot manipulateur et les vitesses associées. Les résultats obtenus dans [7] répondent presque à notre problématique, reprenant une stratégie de planification de trajectoire similaire à [6] en y ajoutant la notion d'évitement d'obstacles dynamiques. Ils ne traitent pas néanmoins du cas d'un véhicule mobile, se concentrant plutôt sur un robot manipulateur. De même, la méthode analytique de planification de chemin dite *model based* développée dans [8] est très intéressante mais le formalisme employé est beaucoup moins flexible que celui du problème d'optimisation du *MPC*.

Il apparait donc que l'utilisation du *MPC* pour la génération de trajectoires a déjà été étudiée, ainsi que plusieurs autres approches équivalentes. Cependant, très peu de travaux semblent traiter spécifiquement de la génération de trajectoires en ligne pour des robots mobiles dans le contexte de l'évitement d'obstacles dynamiques avec le formalisme caractéristique du *MPC*. Il est donc pertinent de se demander si celle-ci est viable et s'il peut présenter un intérêt par rapport aux solutions existantes.

1.4 Méthodologie et organisation

Le projet a été divisé en deux grandes phases, le développement d'une stratégie d'évitement d'obstacles statiques et le développement d'une stratégie d'évitement d'obstacles dynamiques. Une étape intermédiaire centrée sur le développement d'algorithmes de génération de trajectoires hors ligne et en ligne ont également été définies pour la première et la seconde phase du projet respectivement. La mission considérée servant de toile de fond pour l'ensemble du projet a été adapté d'une des missions traitées dans la thèse de PORÉ. Elle décrit un robot évoluant dans un environnement encombré, comprenant des obstacles statiques et dynamiques, devant rallier un nombre arbitraire de points de passages quelconques en évitant les collisions.

La même méthodologie a été prévue pour les deux étapes du projet. Des algorithmes de génération de trajectoires sont d'abord développés et/ou implémentés et testés sur un simulateur basique développé en Python. Une stratégie d'évitement d'obstacles est ensuite dérivée de ces algorithmes et validée sur le même simulateur. Elle est alors implémentée dans un paquet ROS2, afin d'être validée sur la simulation Gazebo officielle du robot Tiago, fournie par PAL Robotics. Elle est enfin déployée sur le robot réel et ajustée pour obtenir les meilleurs résultats possibles.

Outre la simple validation des stratégies, des comparaisons avec des algorithmes classiques ont été réalisées. Les algorithmes A* et RRT*-connect ont été utilisés comme références pour la génération de trajectoires hors ligne et pour l'évitement d'obstacles statiques. La génération en ligne et l'évitement d'obstacles dynamiques ont quant à eux été comparés à un contrôleur *MPC* équivalent.

Il a donc été nécessaire de développer une panoplie d'outils pour simuler convenablement les scénarios de validation et pour effectuer les comparaisons. Le simulateur basique, les obstacles statiques, un contrôleur PID, un contrôleur prédictif et un filtre de Kalman étendu (*EKF*) ont entre autres été implémentés pendant la première étape. Pour leur part, les obstacles dynamiques et l'unité de suivi d'obstacles ont été implémentés durant la seconde étape.

Afin d'essayer d'assurer la bonne marche du projet, une planification à long terme a été mise en place, doublée d'un suivi hebdomadaire de l'avancement. Le détail de la planification à long terme est donné dans le diagramme de Gantt de l'Annexe C-1 et le déroulement réel des activités dans celui de l'Annexe C-2. La planification ne couvre pas les activités des deux premières semaines, qui ont été consacrées à la prise de connaissance du projet et de l'environnement de travail. Le retard accumulé par rapport à la planification prévue est en grande partie explicable par une erreur d'estimation du volume de travail qu'a représenté la première phase du projet. En effet, la majorité des outils ont dû être développés dès la première phase d'optimisation et n'ont eu qu'à être légèrement modifiés pendant les deuxièmes phases d'implémentation et d'optimisation. Les autres difficultés notables sont détaillées dans la section *3.3. Difficultés notables*.

1.5 Buts et enjeux

Dans un premier temps, ce stage avait plusieurs objectifs liés à mon parcours académique. En tant que projet de fin d'études, il devait d'abord me permettre de valider les connaissances que j'ai acquises tout au long de mon cursus en les mettant en application dans le cadre d'un projet de recherche concret. Dans le contexte de mon master, il devait aussi être une occasion pour moi de me familiariser plus encore avec le milieu de la recherche. Compte tenu de mon ambition de faire une thèse, ce stage devait enfin me permettre d'identifier un domaine spécifique de la robotique dans lequel me spécialiser. Ces trois objectifs ont bien été atteints. De fait, j'ai pu mobiliser une grande partie des connaissances que j'ai assimilées à l'*ENSTA* sur un projet de concret dans un laboratoire reconnu. J'ai également pu confirmer mon appétence pour la commande de manipulateurs mobiles dans des environnements dynamiques et même décrocher une thèse dans ce domaine.

Dans un second temps, le sujet présentait plusieurs enjeux pour l'équipe *RAP*. Mes travaux devaient en effet permettre l'amélioration de la génération de trajectoires pour la base du manipulateur mobile utilisée dans le projet de thèse auquel ce stage était rattaché. Ils devaient aussi être utilisés de pair avec d'autres projets traités en parallèle de mon stage au sein de l'équipe pour produire une publication. Ces objectifs ont été partiellement atteints. De fait, les algorithmes de génération de trajectoires que j'ai développés ont été réutilisés par PORÉ dans sa thèse. Cependant, il a été décidé d'attendre le début de ma thèse pour compléter le travail réalisé avant d'envisager une publication.

2 Activités

2.1 Modélisation

2.1.1 Véhicules

On présente dans cette section les modèles utilisés pour tous les systèmes étudiés. Les phénomènes dynamiques pouvant être négligés en première approximation dans les applications visées, un modèle cinématique a été utilisé. $\forall t \in \mathbb{R}_+^*$, le vecteur d'état de la base mobile (resp. du manipulateur) à l'instant t est noté $X_{base}(t)$ (resp. $X_{bras}(t)$). De même, le vecteur d'entrée de la base mobile (resp. du manipulateur) à l'instant t est noté $u_{base}(t)$ (resp. $u_{bras}(t)$). On notera $X_{manip}(t) = (X_{base}(t), X_{bras}(t))$ le vecteur d'état du manipulateur mobile complet à l'instant t et $u_{manip}(t) = (u_{base}(t), u_{bras}(t))$ le vecteur d'entrée du système à l'instant t. On notera enfin $x_{outil}(t)$ et $y_{outil}(t)$ les coordonnées de l'outil du manipulateur mobile dans le repère du monde.

2.1.1.1 Base mobile

Deux types de bases mobiles sont considérées, celles de type Ackermann et celles de type différentielle. On considère le même état $X_{base} = (x, y, \theta)$ pour les deux types de bases, illustré sur la figure 1. Le même modèle cinématique est utilisé pour les deux types de bases mobiles, seuls leurs modèles d'actionnement différent.

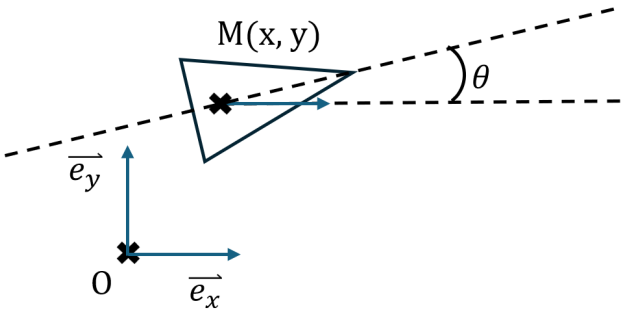


Figure 1 – *Représentation de l'état d'une base mobile*

Un modèle de tricycle est utilisé pour la base mobile de type Ackermann. Il est actionné par la vitesse angulaire moyenne ω_d en entrée du différentiel et par l'angle de la roue directrice équivalente sur l'axe

principal de la base mobile, noté δ. On pose donc $u_{base} = (\omega_d, \delta)$. Le modèle complet est donné dans l'équation 1. Il est décomposé en un modèle d'actionnement (équation 1a) et un modèle cinématique 1b. Ses paramètres géométriques et ses entrées sont illustrées sur la figure 2. La taille de la base mobile de type Ackermann est définie par la formule $L_{base} = \max(1, 9\,L_u, L)$.

$$\begin{pmatrix} v \\ \omega \end{pmatrix} = \begin{pmatrix} \frac{\omega_d \cdot R}{Kr} \\ \frac{\omega_d \cdot R}{Kr \cdot L_u} \tan(\delta) \end{pmatrix} \tag{1a}$$

$$\dot{X} = \begin{pmatrix} v\cos(\theta) \\ v\sin(\theta) \\ \omega \end{pmatrix} \tag{1b}$$

avec :

— $R = 0.075\ m$: le rayon des roues
— $K_r = 0.8$: le coefficient de réduction du différentiel
— $L_u = 1.2\,L$: la distance entre les essieux de la base mobile
— $L = 0.25\ m$: la moitié de la longueur des essieux

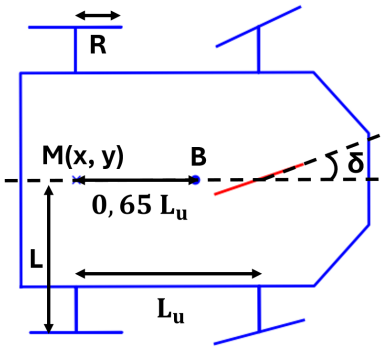


Figure 2 – *Représentation géométrique d'une base mobile de type Ackermann*

Un modèle de base différentielle est utilisé pour la base mobile éponyme. Il est actionné par les vitesses angulaires des roues motrices gauche et droite, notées ω_g et ω_d respectivement. On pose donc $u_{base} = (\omega_g, \omega_d)$. Le modèle complet est donné dans l'équation 2. Il est décomposé en un modèle d'actionnement (équation 2a) et un modèle cinématique 2b. Ses paramètres géométriques et ses entrées sont illustrées sur la figure 3. La taille L_{base} de la base mobile de type différentielle est définie par son diamètre.

$$\begin{pmatrix} v \\ \omega \end{pmatrix} = \begin{pmatrix} \frac{R}{2} & \frac{R}{2} \\ -\frac{R}{L} & \frac{R}{L} \end{pmatrix} \cdot u_{base} \tag{2a}$$

$$\dot{X} = \begin{pmatrix} v\cos(\theta) \\ v\sin(\theta) \\ \omega \end{pmatrix} \tag{2b}$$

avec :

— $R = 0.0985\ m$: le rayon des roues
— $L = 0.2022\ m$: la moitié de la longueur de l'essieu

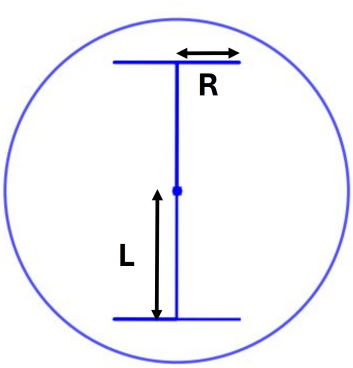


Figure 3 – *Représentation géométrique d'une base mobile de type différentielle*

2.1.1.2 Manipulateur

Un bras manipulateur de $n_{manip} \in \mathbb{N}^*$ segments a été initialement considéré. Ces derniers pouvaient être reliés par des liaisons pivot. Le bras était décrit intégralement par ses degrés de liberté, c'est-à-dire les angles entre les segments. Le manipulateur était piloté par les vitesses angulaires des différentes liaisons, on avait donc :

$$\dot{X}_{manip} = u_{manip} = \begin{pmatrix} \dot{q}_1 \\ \vdots \\ \dot{q}_{n_{manip}} \end{pmatrix} \tag{3}$$

avec :

— q_0 l'angle entre le premier segment du manipulateur et l'axe principal de la base mobile
— $\forall i \in [1, n], q_i$: l'angle du i-ème segment avec le $i-1$-ème segment du manipulateur
— $\forall i \in [1, n], \dot{q}_i$: la vitesse angulaire du i-ème segment

Conformément à la définition du vecteur d'état du système donnée plus haut, la position du manipulateur mobile était définie uniquement par la position de la base mobile. Une transformation homogène représentant la pose du bras dans le repère de la base mobile était alors définie afin de pouvoir convenablement représenter le manipulateur mobile dans son ensemble. Cette transformation constante était considérée comme un paramètre géométrique du manipulateur mobile.

2.1.2 Obstacles

Considérons un obstacle statique $\mathcal{O}$, celui-ci est caractérisé par sa forme circulaire ou rectangulaire. Il est également caractérisé par sa taille, définie par son diamètre $d_{\mathcal{O}}$ s'il est circulaire et par le couple $(l_{\mathcal{O}}, h_{\mathcal{O}}) \in {\mathbb{R}_+^*}^2$ de sa longueur et de sa hauteur s'il est rectangulaire. Il est repéré par sa pose $X_{\mathcal{O}} = (x_{\mathcal{O}}, y_{\mathcal{O}}, \theta_{\mathcal{O}}, v_{x,\mathcal{O}}, v_{y,\mathcal{O}}, \omega_{\mathcal{O}})$, avec :

— $(x_{\mathcal{O}}, y_{\mathcal{O}}) \in \mathbb{R}^2$: la position de l'obstacle dans le repère du monde
— $\theta_{\mathcal{O}} \in [0, 2\pi]$: l'orientation de l'obstacle dans le repère du monde
— $(v_{x,\mathcal{O}}, v_{y,\mathcal{O}}) \in \mathbb{R}^2$: les vitesses linéaires de l'obstacle dans le repère du monde
— $\omega_{\mathcal{O}} \in \mathbb{R}$: la vitesse angulaire de l'obstacle dans le repère du monde

Un obstacle dynamique $\mathcal{O}_d$ possède toutes ces propriétés et est en plus caractérisé par un comportement cyclique, aléatoire ou agressif, permettant de caractériser son mouvement. Un comportement cyclique est caractérisé par le suivi d'un parcours défini par une liste de points de passages $W_{\mathcal{O}_d} = (w_k)_{k \in [\![0, n_{\mathcal{O}_d}]\!]} \in (\mathbb{R}^2)^{n_{\mathcal{O}_d}+1}$, avec $n_{\mathcal{O}_d} \in \mathbb{N}$ / $n_{\mathcal{O}_d} \geq 3$, possiblement différents de ceux utilisés pour la planification de chemin. Notons $n_{split} = \left\lceil \frac{n_{\mathcal{O}_d}}{2} \right\rceil$, la liste $W_{\mathcal{O}_d}$ est alors découpée en $W^a_{\mathcal{O}_d} = (w_k)_{k \in [\![0, n_{split}]\!]}$ et $W^b_{\mathcal{O}_d} = (w_{n_{split}}, \dots, w_{n_{\mathcal{O}_d}}, w_0)$. La méthode présentée dans la section *2.2.1.1. Approche initiale* est utilisée pour générer deux courbes paramétriques $S^a_{W_{\mathcal{O}_d}} : [0,1] \to \mathbb{R}^2$ et $S^b_{W_{\mathcal{O}_d}} : [0,1] \to \mathbb{R}^2$ interpolant les points de passage de $W^a_{\mathcal{O}_d}$ et $W^b_{\mathcal{O}_d}$ respectivement. La trajectoire $\mathcal{T}_{\mathcal{O}_d} : \mathbb{R}_+ \to \mathbb{R}^2$ de l'obstacle cyclique $\mathcal{O}_d$ est ensuite définie par la formule suivante :

$$\forall t \in \mathbb{R}_+, \mathcal{T}_{\mathcal{O}_d}(t) = \begin{cases} S^a_{W_{\mathcal{O}_d}}(2 * p_{mod}(t)) & \text{si } p_{mod}(t) < 0.5 \\ S^b_{W_{\mathcal{O}_d}}(2 * (p_{mod}(t) - 0.5)) & \text{sinon} \end{cases} \tag{4}$$

avec :

— $\forall t \in \mathbb{R}_+^*$, $p_{mod}(t) = fmod(\frac{v_{targ}}{l_{S_W}} t, 1)$: un paramètre permettant de parcourir la boucle formée par $S^a_{W_{\mathcal{O}_d}}$ et $S^b_{W_{\mathcal{O}_d}}$ de façon cyclique.
— $v_{targ} \in \mathbb{R}_+$: la vitesse nominale de l'obstacle (voir section *2.2.2.1. Génération hors ligne*)
— $l_{S_W} \in \mathbb{R}_+$: la longueur de la boucle

Un comportement aléatoire est caractérisé par l'équation d'évolution présentée dans l'équation 5. On note $u_{\mathcal{O}} = (a_{x,\mathcal{O}}, a_{y,\mathcal{O}}, \alpha_{\mathcal{O}})$ les accélérations linéaires et angulaires de l'obstacle aléatoire $\mathcal{O}$. Ces dernières sont obtenues indépendamment à partir de tirages aléatoires dans $\mathbb{R}$ suivant une loi normale centrée d'écart-type égal à 10. Cet obstacle est donc assimilable à un objet évoluant à vitesse constante moyennant un bruit gaussien sur son accélération.

$$\dot{X}_{\mathcal{O}} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} X_{\mathcal{O}} + \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} u_{\mathcal{O}} \tag{5}$$

Un comportement agressif est caractérisé par le même modèle que celui d'un obstacle aléatoire, mais avec un asservissement sur la position de l'obstacle le forçant à poursuivre le robot. L'asservissement étant réalisé en position, l'orientation de l'accélération angulaire de l'obstacle est définie comme pour un obstacle aléatoire. Un correcteur PI est utilisé afin d'assurer l'asservissement. La loi de contrôle correspondante est donnée par :

$$\forall t \in \mathbb{R}_+^*,\ u_{\mathcal{O}} = \begin{pmatrix} (K_p * e_x(t) + K_i * \int_0^t e_x(\tau)d\tau) * \gamma(t) \\ (K_p * e_y(t) + K_i * \int_0^t e_y(\tau)d\tau) * \gamma(t) \\ \alpha_{\mathcal{O}} \end{pmatrix} \tag{6}$$

avec :

— $K_p = 1, K_i = 200$: les gains proportionnel et intégral du correcteur PI
— $e_x(t) = x(t) - x_{\mathcal{O}}(t)$: l'erreur en position sur l'axe des abscisses entre le robot et l'obstacle
— $e_y(t) = y(t) - y_{\mathcal{O}}(t)$: l'erreur en position sur l'axe des ordonnées
— $\gamma(t) = 1 - \min(\max(\sqrt{e_x(t)^2 + e_y(t)^2}/d_{track}, 0), 1)$: un gain de saturation qui permet de réduire la précision du suivi lorsque l'obstacle est proche du robot
— $d_{track} = 1$: le rayon intérieur de la zone de poursuite

Comme expliqué dans la partie *2.2.1. Planification de chemin*, des représentations adaptées à chaque méthode de planification sont dérivées des représentations des obstacles présentées dans cette section. Dans le cas des approches hors ligne, les obstacles dynamiques sont considérés comme statiques à leurs positions initiales. Une zone morte, caractérisée par une distance de padding d_{pad}, est également définie autour de chaque obstacle. Elle a pour vocation d'empêcher la génération d'un chemin passant par un col trop étroit entre deux obstacles. Pour un obstacle circulaire, elle correspond à un cercle de diamètre $d_{\mathcal{O}} + 2\,d_{pad}$ centré sur l'obstacle. Pour un obstacle rectangulaire, elle correspond à un rectangle de longueur $l_{\mathcal{O}} + 2\,d_{pad}$ et de hauteur $h_{\mathcal{O}} + 2\,d_{pad}$ centré sur l'obstacle et orienté selon $\theta_{\mathcal{O}}$. Dans notre cas, on fixe $d_{pad} = 0.55\,L_{base}$ pour toutes les zones mortes mais elles pourraient être ajustées individuellement si besoin. On pourrait imaginer un algorithme de contrôle utilisant des distances de padding plus élevées pour assurer le suivi sous-contraintes de trajectoires obtenues par un algorithme de planification utilisant des distances de padding plus faibles.

2.2 Algorithmes

2.2.1 Planification de chemin

En dehors des approches par optimisation, toutes les méthodes de planification de chemin présentées dans la section *2.2.1. Planification de chemin* ont été utilisées exclusivement hors ligne. On considérera l'ensemble des points de passage $W = (w_k)_{k\in[\![0,n]\!]} \in (\mathbb{R}^2)^{n+1}$, avec $n \in \mathbb{N}^*$, dans la suite du rapport. Dans un effort de standardisation de format entre tous les algorithmes, il a été décidé de représenter les chemins par des listes ordonnées de points de $\mathbb{R}^2$. On notera donc $C_W = (c_k)_{k\in[\![0,m]\!]} \in (\mathbb{R}^2)^{m+1}$ un chemin obtenu à partir de l'ensemble de points de passage W.

2.2.1.1 Approche initiale

La première stratégie de planification développée repose sur une interpolation polynomiale. Ne prenant pas en compte la présence des obstacles, cette méthode sert principalement à générer des trajectoires de références pour le *MPC* qu'il soit utilisé pour du contrôle ou pour de la planification de chemin.

Une expression analytique de la courbe paramétrique $S_W : [0,1] \to \mathbb{R}^2$ interpolant les points de passage de W est d'abord obtenue en utilisant une interpolation de degré $d = \min(card(W), 3)$. un échantillonnage uniforme de la courbe paramétrique est alors réalisé. La liste ordonnée de points $C_W = (c_k)_{k \in [\![0,m]\!]} \in (\mathbb{R}^2)^{m+1}$ obtenue par échantillonnage de S_W est définie par :

$$\forall k \in [\![0,m]\!], c_k = S_W\left(\frac{k}{m}\right) \tag{7}$$

2.2.1.2 Approches par graphe

Une fois le fonctionnement des blocs élémentaires vérifié, d'autres algorithmes de planification plus complexes ont été implémentés. Dans un premier temps plusieurs algorithmes utilisant une approche par graphe (e.g. BFS [9], Dijkstra [10], A* [11], ...) ont été implémentés. Parmi eux, l'algorithme principal retenu a été une variante de l'algorithme A* adapté pour fonctionner sur une grille carrée dont toute case est considérée comme voisine des huit cases adjacentes.

On définit la grille $G \subset (N^2)$ utilisée par l'algorithme A* comme l'ensemble des cases qui la constituent. Chaque case est repérée par sa position $(i,j) \in \mathbb{N}^2$ dans la grille. La case la plus proche de l'origine $O_G \in \mathbb{R}^2$ de la grille est en $(0,0)$ est les coordonnées sont incrémentées de 1 pour chaque pas d'une case vers le haut et vers la droite respectivement. L'origine de la grille étant fixée au niveau de son coin inférieur gauche, la case $(0,0)$ l'est aussi. Dans notre cas, les coordonnées de l'origine O_G sont définies par la formule suivante :

$$O_G = (\min_{p \in W}(p \cdot e_x), \min_{p \in W}(p \cdot e_y)), \text{ avec } (e_x, e_y) \text{ la base canonique de } \mathbb{R}^2 \tag{8}$$

Chaque point de passage $p \in W$ est associé à une unique case de la grille qui le contient en utilisant la formule $round_2((p - O_G)/L_{cell} + 0.5)$, avec L_{cell} la longueur d'un côté d'une case de la grille et où $\forall v(x,y) \in \mathbb{R}^2$, $round_2(v) = (round(x), round(y))$. Une case est considérée comme interdite si son centre est obstrué par un obstacle ou s'il se trouve dans la zone morte d'un obstacle. $\forall k \in [\![0, n-1]\!]$, l'algorithme A* est utilisé pour trouver le chemin $C_{W,k}$ reliant les points de passage w_k et w_{k+1}. Le chemin C_W est alors reconstruit en concaténant les chemins $C_{W,k}$. Notons qu'il est alors obtenu directement sous forme de liste ordonnée de points correspondant aux coordonnées des centres des cases de la grille formant le chemin entre les cases associées à w_0 et à w_n respectivement.

2.2.1.3 Approches par échantillonnage

Dans un deuxième temps d'autres algorithmes utilisant une approche par échantillonnage (e.g. RRT* [12], Informed RRT* [13], RRT*-Connect [14], ...) ont été implémentés. L'algorithme RRT*-connect a été retenu comme algorithme principal de planification par échantillonnage. Il pouvait être réglé pour fonctionner en mode « standard» ou en mode « performance». Le mode standard impose un nombre minimum d'itérations de l'algorithme avant de renvoyer un chemin, ce qui permet d'obtenir un chemin de meilleure qualité, là où le mode performance permet de renvoyer le premier chemin trouvé.

La modélisation utilisée par la simulation et celle utilisée par l'algorithme étant très similaires, presque aucun ajustement n'a été requis. Comme pour l'approche par graphe, on utilise l'algorithme sur chaque paire de points de passage consécutifs puis on concatène les chemins obtenus pour obtention du chemin final C_W. Pour cela, $\forall k \in [\![0, n-1]\!]$, on place le nœud racine de l'un des deux arbres en w_k et celui de l'arbre restant w_{k+1} avant d'exécuter l'algorithme. Le chemin $C_{W,k}$ obtenu est renvoyé sous forme de liste ordonnée de points correspondant aux nœuds des branches formant le chemin le plus court entre les nœuds associés à w_k et à w_{k+1} respectivement. Cependant, il n'est pas utilisé en tant que tel pour reconstruire le chemin C_W. De fait, les branches constituant le constituant sont interpolées par des segments puis échantillonnées à intervalles réguliers. Cette étape a été rajoutée afin de permettre l'utilisation de branches plus longues et donc d'assurer de meilleures performances. On notera que ce processus aurait aussi pu être utilisé pour les chemins obtenus avec des approches par graphe bien que cela n'ait pas été fait.

2.2.1.4 Approches par optimisation

Deux algorithmes ont été développés pour la planification de chemin par optimisation. Contrairement aux approches précédentes, ces algorithmes nécessitent la connaissance d'un modèle du système en plus de la connaissance de l'environnement pour fonctionner. Ils peuvent en contrepartie être réglés pour générer des chemins adaptés spécifiquement à un manipulateur mobile ou à une base mobile, selon la fonction de coût choisie.

Dans les faits, les algorithmes présentés permettent la génération de trajectoires à valeurs dans l'espace d'état du système. Ces trajectoires étant déjà échantillonnées, elles sont implicitement converties en un chemin C_X en omettant la dépendance temporelle. Si le système étudié est la base mobile seule, le chemin C_W est obtenu directement par projection de C_X dans le plan. Si le système étudié est le manipulateur mobile, le chemin C_W est obtenu à en calculant les positions de l'outil du manipulateur par cinématique directe pour chaque point de C_X.

Il est à noter qu'il aurait été préférable ici de chercher à conserver l'information temporelle des trajectoires plutôt que d'effectuer une double conversion trajectoire-chemin chemin-trajectoire. De fait, les trajectoires générées par optimisation n'étant pas nécessairement obtenues à vitesse constante, le balayage indiciel à vitesse constante peut entraîner l'apparition de mouvements incompatibles avec le modèle du système utilisé pour la génération. Cette double conversion a néanmoins été conservée pour faciliter l'étape de filtrage présentée plus bas

On considère les problèmes discrétisés dans cette partie, pour simplifier les notations. On aura donc pour toute fonction f du temps et pour tout $k \in \mathbb{N}$ la fonction discrétisée correspondante $f_d : k \mapsto f(k \cdot dt)$ que l'on confondra avec f. La période d'échantillonnage dt est égale à la période de la boucle de contrôle du système. C'est donc aussi la fréquence temporelle à laquelle les trajectoires générées sont échantillonnées.

Le premier algorithme de planification par optimisation repose sur l'utilisation du contrôleur présenté dans la section *2.2.4.2. Commande prédictive*. La mission est simulée en utilisant l'algorithme décrit dans la partie *2.2.1.1. Approche initiale* pour obtenir le chemin C_W. Une trajectoire de référence $\mathcal{T}_W$ est obtenue à partir de C_W en suivant la procédure décrite dans la partie *2.2.2.1. Génération hors ligne*. Le point suivi est donc placé au niveau de w_0 et sa trajectoire est enregistrée jusqu'à ce qu'il atteigne le point de passage w_n. Ce point correspond au centre de la base mobile ou à l'outil du manipulateur mobile selon le cas d'utilisation. Une sécurité est également implémentée pour arrêter la simulation si le système n'atteint pas le point de passage w_n dans un délai de $t_{max} = 2\,\frac{l_{C_W}}{v_{targ}}$ secondes, la trajectoire partielle obtenue est alors utilisée. On parlera d'« optimisation en amont» pour se référer à cet algorithme dans la suite du rapport.

Le second algorithme repose sur la résolution de problèmes d'optimisation liés par leurs bords pour générer un chemin. La même trajectoire de référence que pour l'optimisation en amont est utilisée. La fonction de coût J à minimiser est la même que celle utilisée dans le contrôleur de la section *2.2.4.2. Commande prédictive*, à la valeur près des coefficients. La principale différence est que le problème d'optimisation est exprimé en termes d'état en non plus de commande. Notons $X(k)$ et $u(k)$ respectivement l'état et les entrées du système, que celui-ci soit le manipulateur mobile ou sa base mobile seule. On parlera d'« optimisations adjacentes» pour se référer à cet algorithme dans la suite du rapport.

$$X(k) = \underset{X(k)}{\operatorname{argmin}}(J(\tilde{X}(k), u(k), k)) \tag{9}$$

avec :

— $J(\tilde{X}, u, k)$: la fonction de coût à minimiser
— $\tilde{X} = (X - X_{ref})(k)$: l'erreur d'état
— $X_{ref}(k)$: l'état de la référence (voir section *2.2.2.3. Trajectoires de référence*)

Comme pour l'algorithme précédent, on considère un horizon temporel d'au plus t_{max} secondes, soit encore un intervalle de $N_{max} = \frac{t_{max}}{dt}$ pas de temps. Cet intervalle est subdivisé en horizons de $N_{segment}$ pas de temps, sur lesquels on cherche alors à minimiser la fonction de coût. Les problèmes sont traités dans l'ordre chronologique. La dernière position de chaque segment de trajectoire obtenu peut ainsi être utilisé comme condition initiale pour déterminer le segment suivant, assurant ainsi la continuité de la trajectoire. Une sécurité est ajoutée pour étendre l'horizon temporel considéré de $N_{segment}$ pas de temps si le point final de la trajectoire de référence sur l'horizon actuel est obstrué par un obstacle.

Les chemins obtenus par optimisation peuvent présenter des irrégularités, surtout ceux générés par optimisation en amont. En effet, ces derniers peuvent contenir des oscillations résiduelles du passage de la référence utilisée pour les générer au travers d'un gros obstacle. Le contrôleur par *MPC* aura alors souvent tendance à faire osciller le système sur place en attendant que la référence soit à nouveau atteignable. Un algorithme de filtrage, détaillé dans l'Annexe D-2 a été implémenté pour permettre d'éliminer ces irrégularités. Cet algorithme a été pensé pour les chemins adaptés aux bases mobiles et il opère sur ces chemins avant qu'ils ne soient projetés dans le plan.

Un seuil de distance arbitraire $d_{min} \in \mathbb{R}_+^*$ en dessous duquel on considère que deux points sont globalement confondus est défini. Une taille indicielle minimale $m_{min} \in \mathbb{N}^*$ et maximale $m_{max} \in \mathbb{N}^*$ sont également définies, ainsi qu'un seuil de variation de cap $\Delta\theta_{max} \in \mathbb{R}_+^*$. Pour chaque point $c_i = (x_i, y_i, \theta_i)$ du chemin C_W, on vérifie si le point $c_j = (x_j, y_j, \theta_j)$, avec $i + m_{min} \leq j \leq i + m_{max}$, clôt une boucle en c_i, c'est-à-dire si $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} < d_{min}$ et si l'écart-type du cap des points de la boucle potentielle est inférieur à $\Delta\theta_{max}$. Si plusieurs boucles sont identifiées, on retient uniquement celle assodié à l'indice j le plus élevé. On prend supprime alors tous les points entre c_i et c_j.

2.2.2 Génération de trajectoires

2.2.2.1 Génération hors ligne

L'intégralité des trajectoires générées sont obtenues à partir des chemins produits par les algorithmes présentés dans la partie *2.2.1. Planification de chemin*. Ces derniers étant décrits par des listes ordonnées de points du plan, ils peuvent facilement être transformés en trajectoire hors ligne. Pour cela, il suffit de les faire parcourir par une cible évoluant à vitesse indicielle constante. En reprenant les notations établies plus haut, la trajectoire associée à C_W, notée $\mathcal{T}_W : \mathbb{R}_+ \to C_W$, est donc définie par :

$$\forall t \in \mathbb{R}_+, \mathcal{T}(t) = c_{i_C(t)}, \text{ avec } i_C(t) = round\left(\frac{m}{t_{cross}} \cdot t\right) \tag{10}$$

avec :

— $t_{cross} = \frac{l_{C_W}}{v_{targ}}$: la durée nominale pour que la cible parcoure le chemin C_W
— $v_{targ} \in \mathbb{R}_+$: la vitesse nominale de la cible
— $l_{C_W} = \sum_{k=0}^{m-1} \|c_{k+1} - c_k\|_2$: la longueur du chemin C_W

Il est à noter que ce choix n'entraîne aucune garantie sur la vitesse instantanée de la cible dans son mouvement curviligne. La seule information sur v_{targ} fournie par cette approche est un ordre de grandeur de la vitesse nominale de la cible. Dans le cas des trajectoires générées par les algorithmes A* et RRT*-Connect, la vitesse de la cible est bien constante égale à v_{targ} puisque les chemins traités sont des lignes brisées échantillonnées à intervalles réguliers. Elle n'est en revanche pas constante dans le cas de l'interpolation bicubique et des algorithmes d'optimisation, par nature des échantillonnages utilisés pour produire des chemins. Ce comportement est largement acceptable et même appréciable dans la mesure où il entraine un ralentissement naturel de la cible dans les sections à forte courbure, facilitant ainsi le suivi de trajectoire par le robot.

2.2.2.2 Génération en ligne

Une autre limite plus problématique de cette approche est qu'elle n'assure pas intrinsèquement le passage par les tous les points de passages. En effet, Si le robot commence trop loin des premiers points de passage, il est possible qu'il ne puisse pas rejoindre la cible avant qu'elle ne les ait dépassés. Afin d'assurer le passage du robot par l'ensemble des points de passages, une condition est ajoutée pour que la cible s'arrête à chaque point de passage jusqu'à ce que le robot l'ait rejoint. Une exception est faite si la cible est prisonnier d'un obstacle au point de passage actuel.

Les algorithmes ne garantissant pas $W \subseteq C_W$, on cherche à définir une fonction associant à chaque point de passage $p \in W$ le point de C_W correspondant. Toute la difficulté de l'association réside dans la nature de cette « correspondance ». En effet, la possible présence de boucles dans le chemin C_W et la résolution arbitraire utilisée pour son échantillonnage rendent cette association non triviale, d'où le développement de l'algorithme fourni dans l'Annexe D-1. Pour chaque point de passage $p \in W$, ce dernier trie les points de C_W par distance croissante à p et conserve uniquement ceux dans un cercle de rayon égal à la moitié la plus grande distance entre deux points consécutifs de C_W. Il trie ensuite les points restants par indice croissant dans C_W et conserve le plus petit indice qui n'est pas déjà attribué.

Avec cette association, on peut alors découper la trajectoire en une succession de segments reliant de points de passage consécutifs en forçant la cible à s'arrêter à chaque point de passage, suivant les conditions définies plus haut. La cible est considérée comme étant au point de passage p si elle est au point de C_W associé à p. L'expression modifiée de la trajectoire $\mathcal{T}_{hybrid} : \mathbb{R}_+ \to C_W$ est la suivante :

$$\forall t \in \mathbb{R}_+, \mathcal{T}_{hybrid}(t) = c_{i_{C_h}(t)}, \text{ avec } i_{C_h}(t) = \begin{cases} \mathcal{T}(t_{freeze}(t) - T_{freeze}(t)) & \text{si la cible est à l'arrêt} \\ \mathcal{T}(t - T_{freeze}(t)) & \text{sinon} \end{cases} \tag{11}$$

avec :

— $t_{freeze}(t)$: l'instant marquant le début du dernier arrêt en date de la cible, à l'instant t
— $T_{freeze}(t)$: le temps total d'arrêt de la cible, tout point de passage confondu, à l'instant t

La méthode de génération de trajectoires résultant de cette légère modification n'est techniquement plus une approche en hors ligne car les valeurs de $t_{freeze}(t)$ et $T_{freeze}(t)$ ne peuvent plus être calculées qu'en ligne. Toutefois, celle-ci repose encore sur une planification de chemin hors ligne.

La génération de trajectoires en ligne ne reprend pas cette approche hybride car les trajectoires considérées sont générés sur des horizons souvent plus courts que le temps nécessaire pour atteindre le dernier point de passage. Il devient alors beaucoup plus dûr de réaliser une association entre les points de passage et les points du chemin. Elle se base donc sur la même technique que la génération hors-ligne en mettant à jour le chemin support à une fréquence arbitraire, prise suffisamment élevée pour tenir compte de l'évolution des obstacles dynamiques. Cette mise à jour est faite en considérant l'ensemble de points de passage renvoyé par l'application $W_{online} : \mathbb{R}_+ \to \mathcal{P}(\mathbb{R}^2)$ à l'instant t, avec :

$$\forall t \in \mathbb{R}_+, W_{online}(t) = (w_k)_{k \in [\![i_{next}(t), n]\!]} \tag{12}$$

où :

$$\forall k \in [\![i_{next}(t), n]\!], w_k = \begin{cases} M(t) & \text{si } k = 0 \\ w_{k+i_{next}(t)-1} & \text{sinon} \end{cases}$$

avec :

— $i_{next}(t)$: l'indice du prochain point de passage à atteindre dans W.
— $\forall t \in \mathbb{R}_+^*\ M(t) = (x(t), y(t)) \in \mathbb{R}^2$: les coordonnées du robot à l'instant t

On définit alors le chemin $C_{W_{online}}(t) = (c^o)_{k \in [\![0, m_{online}(t)]\!]} \in (\mathbb{R}^2)^{m_{online}(t)+1}$ de façon analogue à C_W, avec $m_{online}(t) = card(W_{online}(t))$. Notons $\forall t \in \mathbb{R}_+$, $t_{online}(t) \in \mathbb{R}_+$ le dernier instant en date où le chemin est mis à jour, à l'instant t. L'expression de la trajectoire en ligne $\mathcal{T}_{W_{online}}(t) : \mathbb{R}_+ \to C_{W_{online}}$ est alors donnée par la formule présentée dans l'équation 13.

$$\forall t \in \mathbb{R}_+, \mathcal{T}_{W_{online}}(t) = c_{i_{C,o}(t)} \tag{13}$$

avec :

$$\forall t \in \mathbb{R}_+, i_{C,o}(t) = \begin{cases} round\left(\frac{m_{online}(t)}{t_{cross}}(t_{freeze}(t) - T_{freeze}(t) - t_{online}(t))\right) & \text{si la cible est à l'arrêt} \\ round\left(\frac{m_{online}(t)}{t_{cross}}(t - T_{freeze}(t) - t_{online}(t))\right) & \text{sinon} \end{cases}$$

Bien que le formalisme développé ci-dessus soit techniquement adapté à tous les algorithmes de planification de chemin présentés dans la partie *2.2.1. Planification de chemin*, il est à noter que l'approche par optimisation en amont n'a pas été utilisée pour la génération de trajectoires en ligne en raison des temps de calculs trop importants qu'elle nécessite.

2.2.2.3 Trajectoires de référence

On considère le problème discrétisé dans cette partie, échantillonné à la fréquence de la boucle de contrôle. Considérons cette fois-ci une trajectoire $\mathcal{T}_W = (x_\mathcal{T}, y_\mathcal{T})$ ayant aussi bien pu être générée hors ligne qu'en ligne. Notons de nouveau X et u respectivement l'état et les entrées du système, que celui-ci soit le manipulateur mobile ou sa base mobile seule. On associe à la trajectoire $\mathcal{T}_W$ la référence $R : \mathcal{T}_W \mapsto X_{ref}$, caractérisée par l'état $X_{ref} : k \mapsto X_{ref}(k)$ donné dans l'équation 14. Il est de la forme $(x_{ref}(k), y_{ref}(k), \theta_{ref}(k))$ pour une base mobile et de la forme $(x_{ref}(k), y_{ref}(k))$ pour un manipulateur mobile.

$$\forall k \in \mathbb{N}, X_{ref}(k) = \begin{cases} (x_\mathcal{T}(k), y_\mathcal{T}(k), \theta_{ref}(k)) & \text{pour base mobile} \\ (x_\mathcal{T}(k), y_\mathcal{T}(k)) & \text{pour un manipulateur mobile} \end{cases} \tag{14}$$

avec :

$$\forall k \in \mathbb{N},\ \theta_{ref} = \begin{cases} atan2(y_\mathcal{T}(k+2) - y_\mathcal{T}(k), x_{\mathcal{T})}(k+2) - x_{\mathcal{T})}(k)) & \text{si } k = 0 \\ atan2(y_\mathcal{T}(k) - y_\mathcal{T}(k-2), x_{\mathcal{T})}(k) - x_{\mathcal{T})}(k-2)) & \text{si } done(k) \\ atan2(y_\mathcal{T}(k+1) - y_\mathcal{T}(k-1), x_{\mathcal{T})}(k+1) - x_{\mathcal{T})}(k-1)) & \text{sinon} \end{cases}$$

où :

$$done(k) \iff \begin{cases} i_C(k) \geq m & \text{si } \mathcal{T}_W \text{ a été générée hors ligne} \\ i_{C,h}(k) \geq m & \text{si } \mathcal{T}_W \text{ a été générée de façon hybride} \\ i_{C,o}(k) \geq m_{online}(k) & \text{si } \mathcal{T}_W \text{ a été générée en ligne} \end{cases}$$

La référence est en fait une approximation de l'état, ou d'une partie de l'état, d'un système idéal parcourant la trajectoire $\mathcal{T}_W$ au rythme de la cible associée. On parlera de trajectoire de référence pour désigner le couple $(\mathcal{T}_W, X_{ref})$. Suivant cette logique d'approximation du comportement désirée, une commande de référence u_{ref} a aussi été calculée dans le cadre de la commande d'une base mobile. L'utilisation de cette commande est discutée plus en détail dans la section *2.2.4.2. Commande prédictive*. Elle n'existe que si le modèle d'actionnement de la base considérée $A : (v, omega) \mapsto u$ est inversible. S'il l'est bien, on a $u_{ref} = A^{-1}(v_d, \omega_d)$, avec v_d et ω_d les vitesses désirées, présentées dans l'équation 15.

$$\forall k \in \mathbb{N},\ v_d(k) = \begin{cases} 0 & \text{si } done(k) \\ \sqrt{(x_{\mathcal{T}}(k+1) - x_{\mathcal{T}}(k))^2 + (y_{\mathcal{T}}(k+1) - y_{\mathcal{T}}(k))^2} & \text{sinon} \end{cases} \tag{15a}$$

$$\forall k \in \mathbb{N},\ \omega_d(k) \begin{cases} 0 & \text{si } done(k) \\ \theta_{ref,next}(k) - theta_{ref}(k) & \text{sinon} \end{cases} \tag{15b}$$

avec :

$$\forall k \in \mathbb{N},\ \theta_{ref,next}(k) = \begin{cases} atan2(y_{\mathcal{T}}(k+3) - y_{\mathcal{T}}(k+1), x_{\mathcal{T})}(k+3) - x_{\mathcal{T})}(k+1)) & \text{si } k = 0 \\ \theta_{ref}(k) & \text{si } done(k) \\ atan2(y_{\mathcal{T}}(k+1) - y_{\mathcal{T}}(k), x_{\mathcal{T})}(k+1) - x_{\mathcal{T})}(k)) & \text{si } limit\_case_1(k) \\ atan2(y_{\mathcal{T}}(k+2) - y_{\mathcal{T}}(k), x_{\mathcal{T})}(k+2) - x_{\mathcal{T})}(k)) & \text{si } limit\_case_2(k) \end{cases}$$

où :

$$\forall k \in \mathbb{N},\ \begin{cases} limit\_case_1(k) \iff k \neq 0 \text{ et } \overline{done(k)} \text{ et } done(k+1) \\ limit\_case_2(k) \iff k \neq 0 \text{ et } \overline{done(k)} \text{ et } \overline{done(k+1)} \end{cases}$$

2.2.3 Navigation

2.2.3.1 Localisation

La problématique principale du stage étant la génération de trajectoires, la question de la localisation a été évitée sur le simulateur basique en utilisant la position exacte du robot. Elle a été traité pour la première fois après le passage sur Gazebo. La localisation du robot a alors été assurée par l'implémentation de l'Localisation adaptative de Monte Carlo (*AMCL*) fourni par PAL Robotics avec le simulateur Gazebo du robot Tiago et avec le robot réel.

L'*AMCL* est une méthode de localisation probabiliste dans un environnement utilisant un filtre particulaire. Il se distingue de la localisation de Monte Carlo classique par son caractère adaptatif. En effet, le nombre de particules utilisées augmente avec le degré d'incertitude de la localisation.

L'implémentation du simulateur était cependant à peine suffisante pour assurer la localisation à elle seule. Le suivi de trajectoire était possible mais de mauvaise qualité, avec de grosses oscillations même à faible vitesse et en ligne droite. La position exacte du robot a été utilisée pour confirmer que la fréquence variable de la localisation par *AMCL* était bien la cause principale des problèmes rencontrés. Ces problèmes étaient encore plus prononcés avec le robot réel, où la localisation par *AMCL* était instable en plus d'être lente.

Un filtre de Kalman étendu a alors été mis en place pour assurer la continuité de la localisation. L'état complet du robot était estimé. Le modèle de base différentielle présenté dans la section *2.1.1. Véhicules* a été utilisé pour la prédiction de l'état. L'observation était directement assurée par les estimations d'état renvoyées par l'*AMCL*. On rappelle la forme générale des équations de prédiction et de correction d'un filtre de Kalman étendu dans l'équation 16, où l'on considère le problème en temps discret avec $k \in \mathbb{N}^*$.

$$\begin{cases} \text{Prédiction :} \\ \hat{X}(k|k-1) = f(\hat{X}(k-1|k-1), u(k-1)) \\ \Gamma(k|k-1) = F(k-1)\Gamma(k-1|k-1)F(k-1)^T + \Gamma_\alpha(k-1) \\ \\ \text{Correction :} \\ K(k) = \Gamma(k|k-1)H(k)^T(H(k)\Gamma(k|k-1)H(k)^T + \Gamma_\beta(k))^{-1} \\ \hat{X}(k|k) = \hat{X}(k|k-1) + K(k)(Z(k) - h(\hat{X}(k|k-1))) \\ \Gamma(k|k) = (I - K(k)H(k))\Gamma(k|k-1) \end{cases} \tag{16}$$

avec :

— $\hat{X}(k|k-1)$: l'état prédit à l'instant k en fonction des observations jusqu'à l'instant $k-1$
— $\hat{X}(k|k)$: l'état corrigé à l'instant k en fonction des observations jusqu'à l'instant k
— $\Gamma(k|k-1)$: la matrice de covariance de l'état prédit
— $\Gamma(k|k)$: la matrice de covariance de l'état corrigé
— f : le modèle de prédiction non linéaire
— $F(k-1)$: la jacobienne de f évaluée en $(\hat{X}(k-1|k-1), u(k-1))$
— $\Gamma_\alpha(k-1)$: la matrice de covariance du bruit de processus
— $u(k-1)$: la commande appliquée au système à l'instant $k-1$
— $Z(k)$: l'observation à l'instant k
— h : le modèle d'observation non linéaire
— $H(k)$: la jacobienne de h évaluée en $\hat{X}(k|k-1)$
— $\Gamma_\beta(k)$: la matrice de covariance du bruit d'observation
— $K(k)$: le gain de Kalman à l'instant k

2.2.3.2 Gestion des obstacles

Comme pour la localisation, la gestion des obstacles a été initialement laissée de côté et les positions exactes des obstacles ont été utilisées. Elle représente néanmoins un problème complexe même pour des obstacles statiques. Celui-ci peut être découpé en deux grosses étapes, à savoir la détection et le suivi des obstacles. La gestion d'obstacles dynamiques ajoute une étape intermédiaire de classification au processus et ajoute un aspect prédictif au suivi. De nombreuses solutions sont présentées dans litérature pour chacune des étapes de ce processus. [15]

La question d'une gestion plus réaliste des obstacles ne s'étant posée que lors de l'implémentation de l'évitement dynamique par *MPC*, l'implémentation ou l'adaptation des travaux mentionnés ci-dessus n'aurait pas été possible dans le temps imparti. Seules quelques idées clés en ont donc été extraites afin de mettre en place un squelette fonctionnel de gestion des obstacles. Seul le suivi a été pleinement traité dans l'ébauche de solution développée, bien que la classification aurait pu l'être en considérant la vitesse estimée des obstacles. On citera comme idées principales retenues : l'utilisation de filtres de Kalman pour l'estimation de position des obstacles [16] et l'utilisation de contraintes dépendante du temps pour la gestion des obstacles dynamique par *MPC* [17]. Les contraintes associées aux obstacles sont détaillées dans la section *2.2.4.2. Commande prédictive*.

Seules les positions et les vitesses linéaires des obstacles sont prises en compte dans l'état estimé $\hat{X}_{\mathcal{O}}$ par les filtres de Kalman utilisés pour l'estimation de l'état des obstacles. Le modèle d'évolution utilisé est le même que celui des obstacles aléatoires présentés dans la section *2.1.2. Obstacles*, moyennant l'ajout d'un terme de bruit de processus gaussien $\alpha_{\mathcal{O}}(k)$. Le modèle d'observation est quant à lui donné dans l'équation 17, où l'on note $\beta_{\mathcal{O}}(k)$ le bruit d'observation gaussien. On notera que bien qu'un filtre de Kalman classique aurait suffi ici, le filtre de Kalman étendu été implémenté pour la localisation du système a été réutilisé pour des raisons de simplicité.

$$Z(k)_{\mathcal{O}} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \hat{X}_{\mathcal{O}}(k|k-1) + \beta_{\mathcal{O}}(k) \tag{17}$$

Plusieurs choix ont été faits par manque de temps pour explorer plus en profondeur la gestion des obstacles. D'abord, le nombre total d'obstacles était supposé connu à tout instant, évitant ainsi la nécessité de réaliser une association entre chaque observation et un obstacle. Ensuite, les positions mesurées des obstacles sont obtenues par des capteurs virtuels renvoyant la position bruitée d'un obstacle quand celui-ci est assez proche du robot. Ce sont ces mesures qui sont converties en observations pour les filtres de Kalman en charge de l'estimation de l'état des obstacles. Cette conversion étant assurée par un bloc de suivi supervisant l'ensemble des filtres.

2.2.4 Commande

2.2.4.1 Commande basique

Un contrôleur PID a été développé pour assurer le suivi de suivi d'une trajectoire de référence. Considérons une trajectoire de référence $(\mathcal{T}_W, X_{ref})$. Les vitesses désirées v_d et ω_d permettant de tendre vers l'état de référence sont déterminées par une stratégie de guidage basique, présentée dans l'équation 18. Cette dernière consiste concrètement à réduire les erreurs de position et de cap en trouvant priorisant Les entrées à appliquer sont alors déterminées de la même façon que la commande de référence présentée dans la section *2.2.2.3. Trajectoires de référence*. Cela implique donc que seuls les systèmes ayant un modèle d'actionnement inversible peuvent être contrôlés avec cette approche.

$$\forall t \in \mathbb{R}_+^*, \begin{cases} \omega_d = (K_{p,hdg} * e_{hdg}(t) + K_{i,hdg} * \int_0^t e_{hdg}(\tau)d\tau + K_{d,hdg} * \dot{e}_{hdg}(t)) \\ v_d = (K_{p,pos} * e_{pos}(t) + K_{i,pos} * \int_0^t e_{pos}(\tau)d\tau + K_{d,pos} * \dot{e}_{pos}(t)) * \gamma(t) \\ \quad \text{où} \\ e_{hdg}(t) = \theta_{ref}(t) - \theta(t) \\ e_{pos}(t) = (n_{theta}\cdot)) \ \exp(\frac{|\omega_d|}{\omega_n om}) \ \|\tilde{X}\| \end{cases} \tag{18}$$

avec :

- $K_{p,hdg}, K_{i,hdg}, K_{d,hdg}, K_{p,pos}, K_{i,pos}, K_{d,pos}$: les coefficients du contrôleur PID
- $n_{theta} = (\cos(theta), \sin(theta))$: le vecteur unitaire dans la direction du cap actuel du robot
- ω_{nom} : la vitesse angulaire seuil en deçà de laquelle on cherche à prioriser la correction de l'erreur de position plutôt que celle de cap
- $\tilde{X} = (x(t) - x_{ref}(t), y(t) - y_{ref}(t))$: le vecteur d'erreur de position du robot par rapport à la référence

2.2.4.2 Commande prédictive

Une stratégie d'évitement d'obstacles centrée sur un contrôleur *MPC* non-linéaire a été développé. Bien qu'elle soit compatible avec tout type de système, elle a été principalement pensée pour le suivi de trajectoire par la base mobile seule, comme expliqué plus bas. Les modèles d'évolution utilisés sont ceux présentés dans la section *2.1. Modélisation*. On note $(\mathcal{T}_W, X_{ref})$ la trajectoire de référence utilisée. L'horizon de prédiction est fixé à N_{mpc} pas de temps de la boucle de contrôle. On rappelle la formulation générale du problème de commande optimale en temps discret dans l'équation 19.

$$u(k) = \underset{u}{\text{argmin}}(J(\tilde{X}, u, k))(k) \tag{19}$$

La fonction de coût J est définie dans l'équation 20. Elle est composée de trois termes tous de forme quadratique. Le premier pénalise à l'écart $\tilde{X} = X - X_{ref}$ entre une projection de l'état système X et celui de la référence X_{ref}. Le deuxième pénalise l'écart de la commande u à la commande de référence u_{ref}, quand cette dernière existe. Le troisième terme pénalise les commandes engendrant des vitesses trop élevées.

$$J(\tilde{X},u,k) = \begin{cases} \frac{1}{2}\Big[(\sum_{i=k}^{k+N_{mpc}-1} \tilde{X}(i)^T Q_{base}(1)\tilde{X}(i) + u(i)^T R_{base} u(i) + w(i)\cdot w(i)^T) \\ +X(\tilde{N}_{mpc})^T (30\ Q_{base}(N_{mpc}))\tilde{X}(N_{mpc})\Big] \end{cases} \quad \text{si} \quad n = 0$$
$$\begin{cases} \frac{1}{2}\Big[(\sum_{i=k}^{k+N_{mpc}-1} \tilde{X}(i)^T Q_{outil}(1)\tilde{X}(i)) \\ +30\tilde{X}(N_{mpc})^T (30\ Q_{base}(N_{mpc}))\tilde{X}(N_{mpc})\Big] \end{cases} \quad \text{si} \quad n > 0 \tag{20}$$

avec :

— $X(k) = (x(k), y(k), \theta(k))$ pour la base mobile seule et $X(k) = (x_{outil}(k), y_{outil}(k))$ pour le manipulateur mobile.
— $\forall k \in \mathbb{N}^*, \forall i in [k, k+N-1], Q_{base}(i) = 2^{i-1}\ diag(5, 5, 0.3)$: les matrices de pondération de l'erreur d'état pour la base mobile sans manipulateur
— $R_{base} = 0,1 \cdot 2^{i-1}\ I_2$: les matrices de pondération de l'erreur de commande pour le manipulateur mobile
— $\forall k \in \mathbb{N}^*, \forall i in [k, k+N], Q_{outil}(i) = 2^{i-1}\ I_2$: les matrices de pondération de l'erreur d'état pour le manipulateur mobile

La minimisation est effectuée sous les contraintes détaillées dans l'équation 21. On trouve d'abord les contraintes d'évolution du système, faisant intervenir l'équation d'évolution f du système. On observe ensuite les contraintes sur les commandes du système, traduisant le plus souvent les limites des actionneurs. On a enfin les contraintes d'évitement, visant à assurer une distance minimale entre le système et chaque obstacle $\mathcal{O}$ de l'ensemble des obstacles connus $\mathbb{O}$. On notera par ailleurs que ces dernières ont été écrites avec le contrôle de la base mobile en tête et ne garantissent pas de non-collision pour le bras.

$$\begin{cases} X(k+1) = f(X(k), u(k)) & \forall k \in [\![k, k+N_{mpc}-1]\!] \\ u_{min} \leq u(i) \leq u_{max} & \forall i \in [\![k, k+N_{mpc}]\!] \\ d(X(i), X_{\mathcal{O}}(i)) > 1 & \forall i \in [\![k, k+N_{mpc}]\!], \forall \mathcal{O} \in \mathbb{O} \end{cases} \tag{21}$$

où :

$$d(X, P_{\mathcal{O}}) = \begin{cases} \left\| \frac{X - P_{\mathcal{O}}}{0.5\ d_{\mathcal{O}} + d_s} \right\|_2 & \text{si} \quad \mathcal{O} \text{ est un obstacle circulaire} \\ \left\| \left(\frac{\delta_x}{0.5\ l_{\mathcal{O}} + d_s}, \frac{\delta_y}{0.5\ l_{\mathcal{O}} + d_s} \right) \right\|_4 & \text{si} \quad \mathcal{O} \text{ est un obstacle rectangulaire} \end{cases} \tag{22}$$

avec :

— $P(k) = (x(k), y(k))$: la position du système
— $P_{\mathcal{O}}(k) = (x_{\mathcal{O}}(k), y_{\mathcal{O}}(k))$: la position du centre de l'obstacle $\mathcal{O}$
— $\Delta = (\delta_x, \delta_y) = R(\theta_{\mathcal{O}}(k))(P(k) - P_{\mathcal{O}}(k))$ avec $R(\theta_{\mathcal{O}}(k))$ la matrice de rotation d'un angle $\theta_{\mathcal{O}}(k)$
— d_s : la marge de sécurité pour la distance minimale aux obstacles

La norme $\|\cdot\|_4$ est utilisée pour approximer la norme $\|\cdot\|_\infty$. Dans le cas de l'évitement dynamique, les contraintes de distance minimale sont appliquées à chaque pas de temps sur en sur les nouvelles positions estimées des obstacles. Ces dernières sont obtenues en supposant les accélérations des obstacles constantes sur un horizon de prédiction, égales à leur dernière valeur estimée. Cette approche entraîne une augmentation significative du nombre de contraintes, et donc du temps de calcul, comme expliqué dans la section *3.4. Améliorations possibles.*

2.3 Implémentation

2.3.1 Paquet Python

2.3.1.1 Outils

On traite dans cette section des particularités de l'implémentation des différents algorithmes et outils présentés plus haut. Unn schéma de classique de Runge-Kutta à l'ordre 4 est utilisé pour réaliser la majorité des intégrations numériques, ses coefficients sont précisés dans le tableau *1. Tableau de Butcher de la méthode de Runge-Kutta utilisée*. L'intégrateur a été implémenté en utilisant CasADi [18], permettant son utilisation aussi bien pour la définition des contraintes symboliques du problème de commande optimale que pour l'intégration numérique des équations de la simulation Python. En effet, les problèmes d'optimisation ont été implémentés sous Python en utilisant CasADi et ils ont été résolus avec le solveur Ipopt [19].

0				
0.5	0.5			
0.5	0	0.5		
1	0	0	1	
	1/6	1/3	1/3	1/6

Table 1 – *Tableau de Butcher de la méthode de Runge-Kutta utilisée*

Une description plus générale d'un manipulateur en trois dimensions a commencé à être implémentée juste avant le premier passage sur Gazebo. Cette dernière n'a toutefois jamais été testée puisqu'il a été décidé de se concentrer sur la base mobile presque au même moment ou une première version de la nouvelle modélisation a été implémentée.

La librairie Open Motion Planning Library (*OMPL*) [20] n'a pas été utilisée dans ce projet car des implémentations de la plupart des algorithmes de planification de référence avaient déjà étaient réalisées dans le cadre de travaux antérieurs. Elles ont donc été préférées pour des raisons de simplicité. Il serait néanmoins intéressant de standardiser le code en l'intégrant cette librairie dans le futur.

2.3.1.2 Structure

Le code réalisé a été structuré en un paquet Python. Le paquet a été développé et testé sur Python 3.10.12. Les dépendances principales sont numpy (>=1.22.0, <1.25.0), scipy (==1.11.4), casadi (==3.6.7) et Rtree (==1.2.0). L'établissement des versions exactes des dépendances a été assez fastidieux lors du passage sur ROS2. Les difficultés rencontrées sont abordées dans la section *3.3. Difficultés notables.*

Le paquet a été pensé pour pouvoir être utilisé sur des systèmes disposant ou non d'une interface graphique. En plus des dépendances principales, matplotlib (>=3.5.1, <3.7.0) et PyQt5 (==5.15.11) peuvent être installés comme dépendances optionnelles pour l'affichage graphique. Un utilitaire d'installation a été développé sous la forme d'un script bash afin de faciliter le déploiement du code sur une nouvelle machine. Des scénarios de tests ont également été mis à disposition pour tout type d'installation.

Le code est pensé pour être modulaire. Il est donc assez facile d'ajouter de nouveaux objets, comme des robots ou des obstacles, en créant une classe fille de la classe abstraite correspondante. Ce sont ces classes abstraites qui assurent la compatibilité des différents objets avec le reste du code par le biais de méthodes abstraites et d'attributs clés. Des setters et getters ont également été utilisés pour permettre une manipulation plus aisée des instances des différentes classes.

2.3.2 Paquet ROS2

Un paquet ROS2 [21] a été créé pour permettre l'utilisation des algorithmes développés sur le simulateur Gazebo du robot Tiago fourni par PAL Robotics et sur le robot réel. Le robot Tiago utilisé est un manipulateur équipé d'un bras à 7 degrés de liberté en rotation. Un degré de liberté supplémentaire en translation est assuré par le tronc du robot. Sa base mobile différentielle est stabilisée par 4 roues libres et peut être équipée de capteurs lasers SICK-TiM561.

Quatre nœuds sont définis dans le paquet. Les trois nœuds principaux sont le *navigation_node*, le *command_node* et l'*environment_node*, associés respectivement aux fonctions de navigation, de commande et de cartographie. Le *navigation_node* est celui responsable de l'estimation de l'état du robot, il fait tourner le filtre de Kalman étendu présenté dans la section *2.2.3.1. Localisation*. Le *command_node* est celui qui fait tourner le contrôleur utilisé, qu'il s'agisse du contrôleur basique ou du contrôleur prédictif. Le *environment_node* aurait été celui qui gère les obstacles, il aurait fait tourner les filtres de Kalman utilisés pour estimer leurs états. Bien que la trame de ce nœud ait été implémenté, les fonctionnalités de suivi d'obstacles qu'il devait exploiter n'ont pas pu être intégrées dans le paquet ROS2 dans le temps imparti. Un dernier nœud, *basic_sim_node*, permet quant à lui l'utilisation du simulateur Python comme backend à la place de Gazebo. Il à grandement facilité la mise en place de l'architecture ROS2 et le débogage en supprimant le besoin de lancer Gazebo à chaque changement, même mineur, du code.

Le noeud de navigation utilise les les commandes publiées sur le topic «*/cmd_vel*» et estimations renvoyées sur le topic «*/amcl_pose*» par l'*AMCL* pour réaliser la prédiction et la correction du filtre de Kalman. Il publie les estimations obtenues sur le topic «*/navigation/state_estimation*». La commande exploite les estimations publiées sur le topic «*/navigation/state_estimation*» pour calculer les nouvelles commandes à appliquer et les publie sur le topic «*/cmd_vel*». Comme indiqué sur le graph, un topic «*/obstacles/pose_vel*» a été mis en place pour informer la commande et la navigation des états estimées des obstacles Cependant, il n'est pas utilisé dans l'état actuel du paquet. L'architecture du paquet est résumée sur la figure *4. Graphe des nœuds et topics du paquet ROS2*.

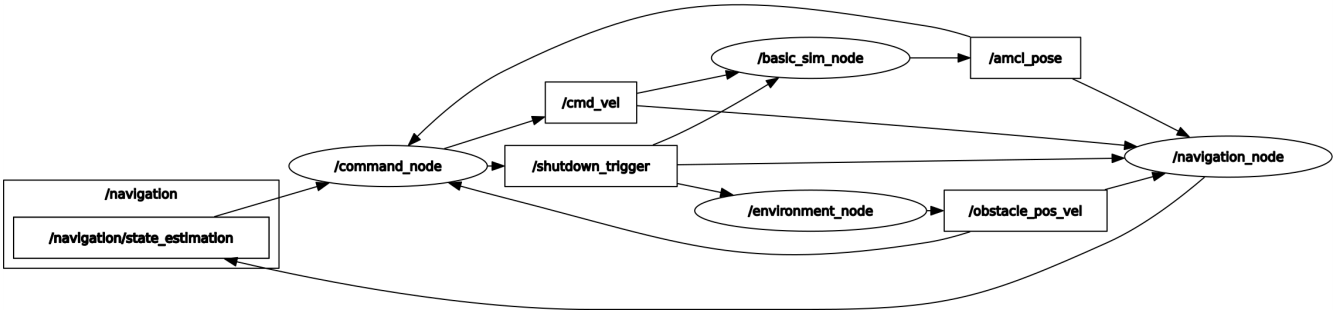


Figure 4 – *Graphe des nœuds et topics du paquet ROS2*

Un fichier de configuration principal a été créé. Celui-ci contient pour l'instant des informations sur les temps de boucle des différents nœuds, sur la graine aléatoire à utiliser dans le code et sur la mission à simuler quand le nœud de simulation basique est utilisé. On pourrait imaginer y ajouter des informations sur les modèles des obstacles ou du robot, comme leurs paramètres géométriques ou cinématiques. Il a pour but d'assurer le partage d'une unique vérité entre tous les nœuds sans avoir à synchroniser les instances dupliquées de chacun d'entre eux via des topics. Les instances de chaque nœud deviennent alors simplement des unités de traitement des informations échangées via les topics.

Deux configurations de lancement ont été établies, une permettant de lancer le pilote automatique composé des nœuds principaux et une autre permettant de lancer le simulateur basique en parallèle du pilote automatique. En raison du temps de lancement trop élevé de Gazebo, la simulation du robot Tiago fournie par PAL doit être lancée manuellement avant le pilote automatique.

2.4 Expérimentations

Les différentes méthodes de génération de trajectoires et d'évitement développées ont été validées et comparées à des algorithmes classiques à chaque étape de leur développement. Une stratégie d'évitement d'obstacles statique et deux stratégies d'évitement d'obstacles dynamiques ont été développées et analysées. L'évitement d'obstacles statique combine la génération d'une trajectoire de référence hors ligne et le suivi de cette trajectoire par un contrôleur basique. La première stratégie d'évitement dynamique fonctionne de la même manière mais avec une trajectoire générée en ligne par optimisations adjacentes. La seconde stratégie d'évitement dynamique utilise un contrôleur *MPC* suivant une trajectoire de référence générée hors ligne par interpolation.

Trois scénarios ont été mis en place pour la validation et la comparaison des algorithmes. Dans le scénario A, le robot est initialement positionné en $(0, 0)$ avec un cap nul et les points de passages sont $W = [(0, 0), (5, 0), (0, 0)]$. Un seul obstacle statique circulaire de $0.5\,m$ de diamètre est dans l'état $X = (2.65, 0, 0, 0, 0, 0)$. Dans le scénario B, le robot est initialement dans le même état que dans le scénario A et les points de passages sont $W = [(0, 2), (-2.5, 2.5), (3.5, 2.5)]$. Deux obstacles statiques rectangulaires de tailes $(1.1, 0.5)$ et $(1.4, 1.4)$ sont dans l'état $(-1.1, 0.84, -\frac{\pi}{3}], 0, 0, 0)$ et $(-0.98, 3.17, 0, 0, 0, 0)$ respectivement. Un obstacle statique circulaire de $0.5\,m$ de diamètre est aussi initialisé dans l'état $(1.66, 2.86, 0, 0, 0, 0)$. Le scénario C est identique au B mais avec un obstacle agressif circulaire additionnel de $0.5\,m$ de diamètre initialement dans l'état $(0, 0, 0, 0.5, 0, 0)$.

2.4.1 Validation des algorithmes

Les parties *2.4.1. Validation des algorithmes* et *2.4.2. Comparaison des algorithmes* se concentrent sur la description des expérimentations réalisées. Les résultats sont présentés dans la section *3.2. Données expérimentales*. Les deux approches par optimisation ont d'abord été validées dans le cadre de la génération hors ligne sur le simulateur Python basique. Les scénarios A et B ont été utilisés pour cette validation. Le scénario A a joué le rôle d'expérience témoin et le scénario B a permis la validation des algorithmes dans des conditions plus réalistes.

L'évitement d'obstacles statiques a ensuite été validé en simulation et en conditions réelles. Les scénarios A et B ont encore une fois été utilisés en simulation mais seul le scénario A a été utilisé en conditions réelles. Les difficultés rencontrées sur le robot réel sont expliquées plus en détail dans la partie *3.3. Difficultés notables*). Les trajectoires de références ont été générées par optimisations adjacentes et en amont ont lors des tests en simulation et des par l'algorithme A* ont lors des tests en conditions réelles. Des obstacles virtuels ont été utilisés pour la validation sur Gazebo et en conditions réelles.

Les deux stratégies d'évitement d'obstacles dynamiques ont été testées sur le simulateur Python basique. Les tests ont été initialement effectués sur le scénario C pour les deux stratégies. Toutefois, les résultats non-concluants obtenus avec la première stratégie ont conduit à la réalisation de tests complémentaires sur le scénario B. Les temps de complétion, temps passés dans les zones dangereuses et distances totales parcourues ont été mesurées pour l'évitement dynamique par contrôleur *MPC*

2.4.2 Comparaison des algorithmes

Une comparaison des temps de génération et des longueurs des trajectoires générées hors ligne a été effectuée sous le simulateur Python basique. Les deux algorithmes d'optimisation ont été comparés avec les algorithmes A*, RRT*-Connect. L'algorithme A* a utilisé une grille avec des cases de $\frac{L_{base}}{5} = 0.108\,m$ de côté, tandis que l'algorithme RRT*-Connect a utilisé pour un nombre fixe de 150 itérations avec des branches de $\frac{L_{base}}{5} m$ de long. L'algorithme de planification par optimisations disjointes a utilisé un horizon de $N_{segment} = 30$ pas de temps de $0.1\,s$ chacun. Le *MPC* utilisé dans la méthode d'optimisation en amont a été réglé pour réaliser des prédictions sur un horizon de $N_{segment} = 30$ pas de temps à une fréquence de $10\,Hz$. L'objectif principal de cette analyse était d'établir un ordre de grandeur de l'efficacité théorique des méthodes développées. Elles ont été réalisées sur les scénarios A et B.

La qualité de l'évitement d'obstacles statiques basé sur ces mêmes algorithmes a ensuite été analysée sur le simulateur basique. Les temps de complétion, temps passés dans les zones dangereuses et distances à la référence ont été mesurés pour chaque méthode. Les algorithmes ont été réglés de la même façon que pour la comparaison de la génération de trajectoires hors ligne. L'objectif principal de ce test est de déterminer si la complexité ajoutée des approches par optimisation se traduit par une amélioration pertinente de la qualité de l'évitement statique. Les scénarios A et B ont encore une fois été utilisés.

3 Résultats

3.1 Travail accompli

Plusieurs algorithmes ont été développés et implémentés dans le cadre de ce projet. En effet, deux algorithmes de génération de trajectoires basés sur des méthodes d'optimisation ont été développés et plusieurs algorithmes classiques ont également été adaptées. Un contrôleur PID simple a été développé pour le suivi de trajectoire et un contrôleur *MPC* pour l'évitement d'obstacles. Un simulateur basique a été créé pour faciliter le développement et le test des divers algorithmes. Plusieurs filtres de Kalman ont été utilisés pour la localisation et le suivi des obstacles. Les prémices d'un processus de détection et de suivi multi-objets a été mis en place. Plusieurs types d'obstacles et de véhicules ont été modélisés.

Le code a été conçu pour être facilement extensible. En outre, Il a été mis sous forme de paquet Python afin de permettre une utilisation et une installation aisées. Par ailleurs, un paquet ROS2 a été créé pour permettre l'intégration avec le simulateur Gazebo et le robot. Les deux paquets sont partiellement documentés et des fichiers README présentant leurs fonctionnalités principales sont disponibles sur les dépôts git associés.

Plusieurs batteries de tests, comparaisons et analyses de performances ont été effectuée en simulation, à la fois sur le simulateur basique et sur Gazebo. Des expérimentations en conditions réelles ont également été réalisées sur le robot. Les résultats obtenus sont présentés dans la section 3.2.

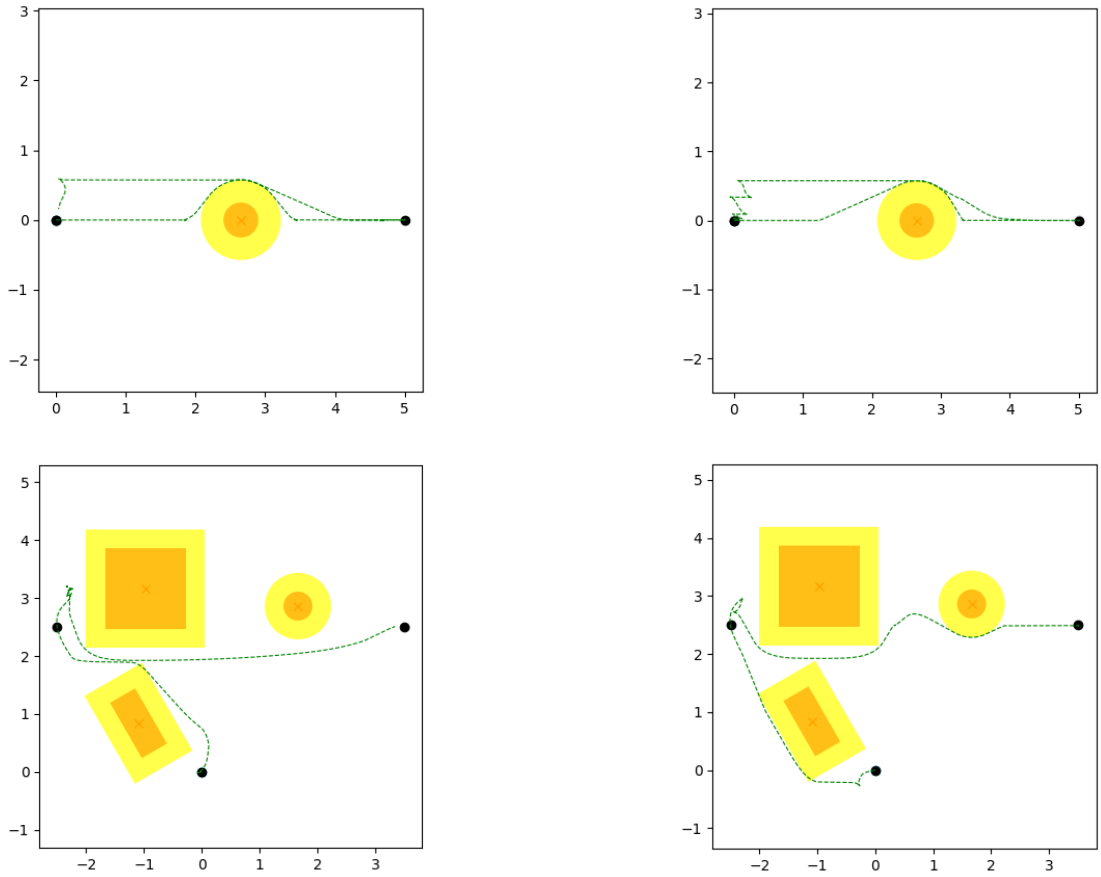
3.2 Données expérimentales

3.2.1 Validation des algorithmes

Les résultats des tests de génération de trajectoires hors ligne sont présentés sur la figure 5. On y constate que les deux approches par optimisation permettent bien de générer des trajectoires respectant toutes les contraintes imposées. Certains comportements indésirables sont toutefois observés.

Dans le scénario A, l'approche par optimisations adjacentes produit une manœuvre en apparence inutile sur la fin de la trajectoire au lieu de rejoindre directement le point de passage final comme le fait l'approche par optimisation en amont. Ce comportement est possiblement dû à un horizon de prédiction trop court entraînant l'apparition d'un piège local. Cette irrégularité bien qu'indésirable était anticipée car la taille de l'horizon de prédiction a été fixé pour essayer de trouver un compromis entre qualité de la trajectoire et temps de calcul. En dehors de cette anomalie, les deux approches fournissent des trajectoires similaires.

Dans le scénario B, c'est l'approche par optimisation en amont qui contient un artefact. On peut en effet observer de légères oscillations après le deuxième point de passage, juste avant l'obstacle le plus gros. Ce comportement a déjà été discuté dans la section 2.2.1.4 et il apparaît ici car les trajectoires présentées n'ont pas été filtrées. L'approche par optimisations adjacentes réussit à générer une manœuvre plus propre pour contourner ce même obstacle. On remarquera par ailleurs que les deux approches produisent ici des trajectoires assez différentes bien que toutes deux valides.



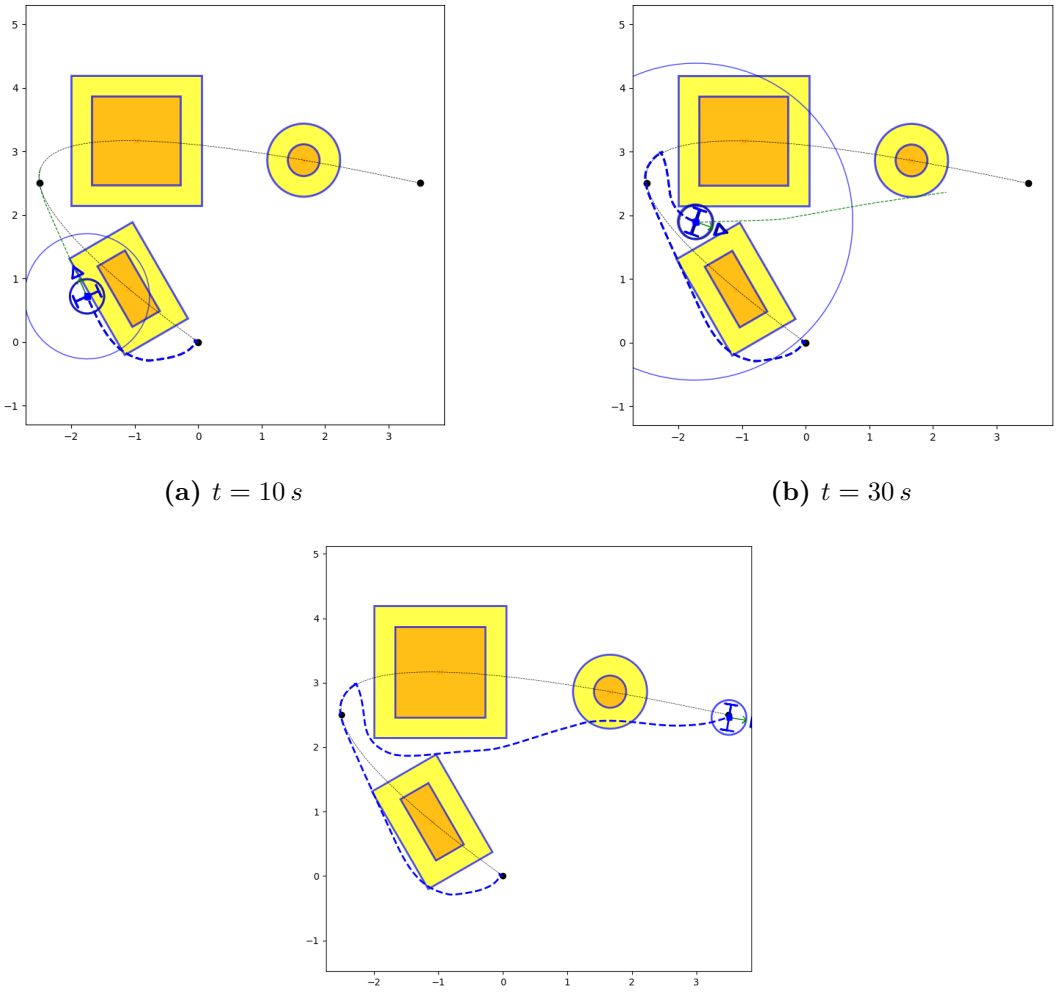
(a) *Approche par optimisation en amont* **(b)** *Approche par optimisations adjacentes*

Figure 5 – *Trajectoires obtenues par génération hors ligne (axes gradués en mètres)*

Pour éviter toute ambigüité entre la trajectoire réelle du système et sa trajectoire de référence, on appellera « trace » sa trajectoire réelle. Les traces obtenues pour l'évitement d'obstacles statiques sont présentées dans l'Annexe E. Les deux approches produisent des résultats partiellement satisfaisants pour chacun des scénarios, sur Python comme sur Gazebo.

Bien que le système parvienne à éviter les obstacles, on observe quelques passages rapides dans la zone de danger des obstacles circulaires. Ce problème est dû au contrôleur qui est volontairement réglé pour être assez souple afin qu'il puisse s'adapter aussi bien à des trajectoires lisses que saccadées. On notera enfin que les irrégularités présentes sur la fin des traces obtenues pour le scénario B sont dues à un obstacle présent dans l'environnement de Gazebo mais absent du simulateur basique. Les scénarios ayant été conçus pour le simulateur basique et étant donné le faible impact sur les résultats, il a été décidé de ne pas modifier l'environnement de Gazebo.

L'évitement d'obstacles statique par génération de trajectoires en ligne réalisé pour le scénario B est illustré en figure 6. On observe un comportement très proche de celui obtenu avec une trajectoire de référence générée hors ligne, confirmant la validité de l'approche. On retrouve malheureusement des défauts similaires avec un empiètement sur la zone de danger des obstacles circulaires.



(a) $t = 10\,s$ **(b)** $t = 30\,s$

(c) $t = 46.35\,s$

Figure 6 – *Trace obtenue pour l'évitement d'obstacles dynamiques basé sur la méthode des optimisations adjacentes (axes gradués en mètres)*

L'évitement d'obstacles dynamiques obtenu avec le contrôleur *MPC* pour le scénario C est présenté sur la figure 7. Le temps de complétion est de 38.80 s, le temps passé dans en zone dangereuse est de 3.70 s et la distance totale parcourue est de 12.43 m. Le test a été relancé une dizaine de fois et les valeurs obtenues n'ont pas montré de variation significative.

Les résultats sont très satisfaisants. En plus de ne quasiment pas empiéter sur la zone de danger des obstacles, le robot parvient à atteindre la référence en un temps raisonnable tout en évitant l'obstacle dynamique. Par ailleurs, on n'observe aucun artefact dans la trace produite. Les performances de l'algorithme ont également été testées sur des versions plus extrêmes du scénario C avec succès, moyennant la nécessité de supprimer les limites de vitesse du modèle du robot.

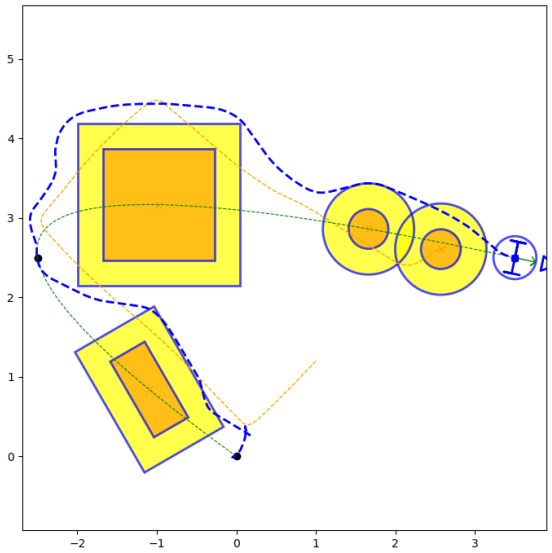


Figure 7 – *Trace obtenue pour l'évitement d'obstacles dynamiques basé sur le contrôleur MPC (axes gradués en mètres)*

3.2.2 Comparaison des algorithmes

Les données relatives à la comparaison des méthodes de génération de trajectoires hors ligne sont présentées dans le tableau 2. D'un côté, les algorithmes classiques (A*, RRT*-Connect) génèrent rapidement des chemins courts dans toutes les situations. De l'autre, les approches par optimisation produisent des chemins plus longs en un temps plus important, avec l'approche par optimisation en amont pouvant atteignant deux ordres de grandeur de temps de calcul de plus que A*. Cette différence s'explique facilement par la prise en compte des contraintes liées au modèle dans les approches par optimisation. On notera de plus que l'algorithme d'optimisation disjointe reste relativement compétitif face à RRT*-Connect, permettant une génération rapide de chemins optimisés lorsque les problèmes sont simples.

Scenario	Algorithme	Temps de calcul (s)	Longueur de la trajectoire (m)
Scénario A	A*	0.06 ± 0.01	11.01
	RRT*-connect	1.93 ± 0.24	10.41 ± 0.01
	Optimisations adjacentes	2.37 ± 0.40	12.28
	Optimisation en amont	12.98 ± 1.28	11.33
Scénario B	A*	0.51 ± 0.01	10.82
	RRT*-connect	3.17 ± 0.07	11.55 ± 0.14
	Optimisations adjacentes	8.01 ± 0.83	12.22
	Optimisation en amont	38.10 ± 0.32	12.36

Table 2 – *Comparaison des caractéristiques des trajectoires générées*

Les données relatives à la comparaison des stratégies d'évitement d'obstacles statiques sont présentées dans le tableau 2. Ils mettent en évidence des comportements contrastés selon les familles d'algorithmes. Dans le scénario A, les méthodes classiques comme A* et RRT*-Connect offrent les meilleures performances en termes de temps de complétion et d'évitement, avec un temps passé en zone de danger, en général, quasiment nul. Dans le scénario B, plus complexe, les résultats sont plus mitigés, avec A* proposant le chemin le plus dangereux. On notera aussi la perte de temps engendrée par l'artefact de la trajectoire générée par l'approche par optimisation en amont observé plus tôt. Ces résultats suggèrent que les approches classiques sont mieux adaptées aux environnements simples et statiques, tandis que les méthodes optimisées présentent un potentiel intéressant pour des contextes dynamiques plus complexes, comme le suggère aussi la figure 7.

Scenario	Algorithme	Temps de complétion (s)	Temps passé en zone de danger (s)	Distance à la référence (m)
Scénario A	A*	39.80	0.00	0.23 ± 0.10
	RRT*-connect	37.50	0.00	0.15 ± 0.08
	Optimisations adjacentes	44.00	4.05	0.17 ± 0.27
	Optimisation en amont	41.10	4.00	0.21 ± 0.19
Scénario B	A*	39.40	3.55	0.23 ± 0.13
	RRT*-connect	40.20	0.90	0.18 ± 0.09
	Optimisations adjacentes	42.80	2.60	0.32 ± 0.14
	Optimisation en amont	46.80	1.15	0.28 ± 0.34

Table 3 – *Comparaison des caractéristiques des traces générées*

Les graphes de l'erreur de position à la référence sont présentés dans l'Annexe F. Cette dernière met en évidence l'une des forces des approches par optimisation. La rapidité des méthodes classiques est en partie due à la faible résolution spatiale des chemins convertis en trajectoires. Cette faible résolution entraîne un mouvement saccadé du système dû à un mouvement saccadé de la référence. Ce n'est pas le cas des approches par optimisation qui produisent des trajectoires globalement plus lisses.

Concernant les caractéristiques distinctives de chaque graphe, on retrouve un pic central dans les graphes des approches classiques sur le scénario A, correspondant au demi-tour effectué au niveau du second point de passage. Celui-ci est réalisé instantanément par la référence, mais le système met un certain temps à s'y adapter. Il est notablement absent des graphes des approches par optimisation sur ce même scénario, remplacé par deux autres pics dues à l'incapacité du système à rattraper la référence en ligne droite. Pour une raison non établie, les références associées aux trajectoires générées par optimisation semblent aller plus vite que ce que le robot est capable de suivre en ligne droite.

On retrouve également deux pics sur les graphes de toutes les approches pour le scénario B. Ils sont dus aux mêmes raisons que ceux observés pour le scénario A, à savoir des virages serrés et des prises de retard pour les méthodes classiques et pour les méthodes par optimisation respectivement. On notera que les pics sont les moins prononcés pour la trajectoire obtenue avec l'algorithme RRT*-Connect. Celui-ci effectue en fait un grand détour pour éviter le premier obstacle.

3.3 Difficultés notables

Les principaux problèmes rencontrés ont été la définition du projet et la gestion du temps. En effet, la problématique a été assez difficile à assimiler malgré une nouvelle direction en apparence assez claire après la réorientation du stage. Mon manque d'expérience avec la commande prédictive et l'exploration de plusieurs sujets connexes ont ajouté à ma confusion. De fait, il m'a été demandé plusieurs fois d'explorer des sujets connexes à la problématique principale, comme la commande partagée, dans le cadre de la potentielle publication mentionnée en introduction. Finalement, c'est moins une unique difficulté qu'une succession de petits problèmes et de mauvaises décisions qui ont entrainé la prise de retard responsable du non-aboutissement du projet.

Une de ces mauvaises décisions a été de trop m'investir sur le simulateur basique. Dans une volonté de légitimer ledit simulateur, tentative de parallélisation pour découplage horloges. Une parallélisation en multithreading a été implémentée. Néanmoins, les threads de Python étant astreints au même processus, des problèmes de performance sont apparus. Une tentative de passage en multiprocessing a été faite. Cependant, les problématiques de mémoire non partagée se sont avérées être trop lourdes à résoudre pour que cette tentative aboutisse. La parallélisation sur le simulateur Python a été complètement abandonnée et laissée à Gazebo.

Un autre problème considérable a été la localisation du robot réel. À cause de ses quatre roues folles, le robot Tiago utilisé présente la particularité de fortement dévier de cap au moment de repartir en ligne droite après un virage sur place. Ce comportement indésirable non modélisé, combiné à une localisation par *AMCL* d'une fréquence moyenne de l'ordre du Hz, a causé des oscillations forte du suivi de trajectoire, présentes notamment sur la figure 8. En raison du retard déjà accumulé sur le projet, il a été décidé de passer à l'implémentation de l'évitement d'obstacles dynamiques plutôt que de tenter de passer sur de la motion capture pour essayer de passer outre les problèmes de localisation.

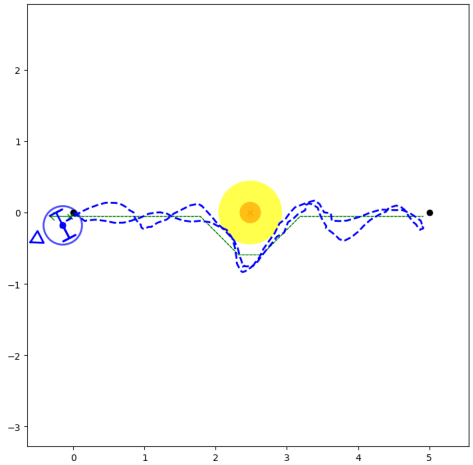


Figure 8 – *Traces obtenue pour l'évitement d'obstacles statiques en condition réelles (axes gradués en mètres)*

Enfin, un problème assez peu impactant mais très étrange a été rencontré lors de la configuration du docker utilisé pour faire tourner Gazebo et le paquet ROS2 développé. Le docker file de base, ayant été utilisé par un autre stagiaire sur sa machine générait une image incomplète sur la mienne. Plus étrange encore, cloner directement l'image de mon collègue depuis DockerHub menait au même résultat. Le problème a été résolu en installant manuellement les dépendances manquantes sur l'image via un container.

3.4 Améliorations possibles

Concernant la génération de trajectoires, les temps de calcul des approches par optimisation doivent être significativement réduits, notamment en présence d'obstacles dynamiques. Une optimisation du calcul symbolique réalisé avec CasADi pourrait être envisagée, plus particulièrement au niveau de l'expression des contraintes utilisées dans le problème d'optimisation. Une analyse sérieuse de l'intégration de certaines contraintes directement dans le modèle pourrait également être intéressante à mener. Un changement de solver pourrait aussi être envisagé. L'utilisation d'Acados [22] de paire avec CasADi semble être une piste intéressante.

Concernant plus spécifiquement la génération de trajectoires en ligne, l'approche par optimisations adjacentes pourrait doit être améliorée. De fait, un des problèmes majeurs de cette technique est qu'elle ne fonctionne pas si la référence est piégée à l'instant final de l'horizon de prédiction. On entends par la que des discontinuités pouvant entrainer des collisions peuvent apparaître, comme illustré en figure 9.

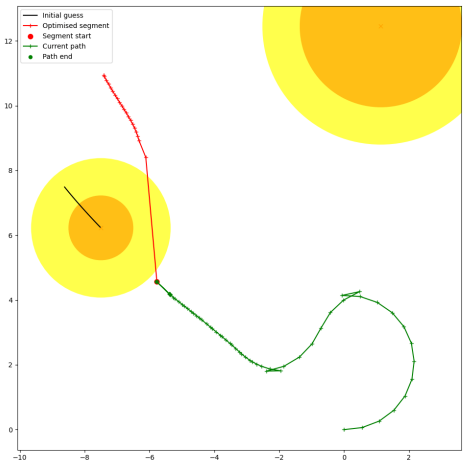


Figure 9 – *Exemple d'horizon de prédiction invalide pour la méthode des optimisations adjacentes*

Dans le cadre de l'évitement d'obstacles statiques, ce problème peut être évité en utilisant un horizon adaptatif et en s'assurant que les premiers et derniers waypoints ne sont pas piégés. Dans ces conditions, on peut s'assurer qu'en étendant suffisamment l'horizon de prédiction en partant d'un point sûr, on finira par atteindre un point sûr. On obtiendra alors un horizon sur lequel le problème d'optimisation est valide, comme illustré en figure 10. Le seul désavantage de cette méthode est que le temps de calcul devient très important quand l'horizon de prédiction est grand.

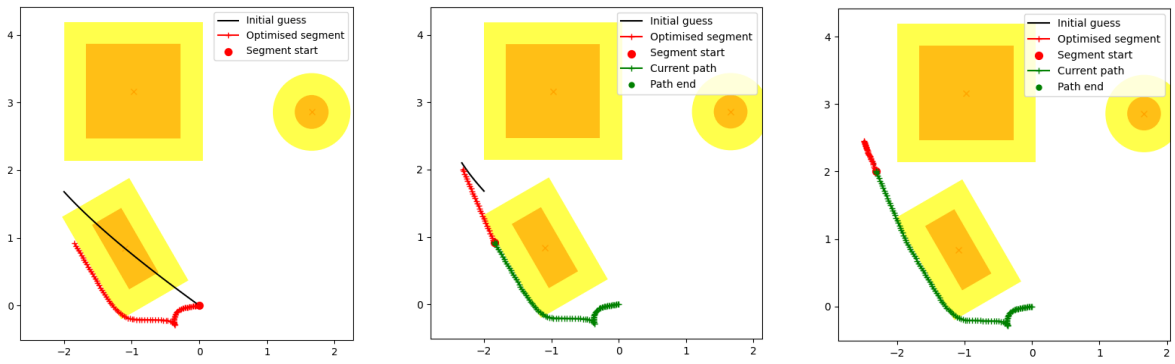


Figure 10 – *Fonctionnement de l'horizon adaptatif utilisé par la méthode des optimisations adjacentes (axes gradués en mètres)*

Cette approche n'est cependant pas viable en présence d'obstacles dynamiques puisque n'importe quel point de l'espace peut devenir piégé à tout instant. Une solution intéressante qui n'a pas pu être explorée est la projection de la référence dans un espace valide. La complexité résiderait dans la définition d'une telle projection en présences de plusieurs obstacles.

Concernant l'évitement d'obstacles, Le contrôleur PID semble avoir besoin d'un léger ajustement pour meilleur les performances du suivi de la référence, comme évoqué dans la section *3.2.1. Validation des algorithmes*. Étant donné le faible empiètement sur la zone de danger, augmenter la zone de danger des obstacles circulaires et rendre le contrôleur plus nerveux semblent être deux solutions viables.

Pour conclure, quelques améliorations secondaires pourraient aussi être apportées au projet. La ré-implémentation du manipulateur mobile dans le travail effectué pourrait d'abord être envisagée. Malgré un abandon du système en milieu de projet, la flexibilité des algorithmes et du code pourrait permettre une réintégration relativement aisée. Il convient enfin de noter que le code pourrait être plus standardisé. Comme évoqué plus tôt, de nombreuses bibliothèques standard comme *OMPL* ou robotics-toolbox pourraient être intégrées pour améliorer les performances et la maintenabilité du code. De plus, la documentation des paquets Python et ROS2 pourrait être étoffée pour compléter convenablement les fichiers README déjà présents.

4 Conclusion

L'objectif principal du projet était d'étudier et de développer des stratégies d'évitement d'obstacles pour un robot mobile évoluant dans un environnement dynamique. Le travail s'est inscrit dans la continuité des recherches menées au sein de l'équipe RAP du LAAS-CNRS.

Plusieurs contributions ont été réalisées au cours de ce stage. Divers algorithmes de génération de trajectoires hors ligne et en ligne ont été développés à partir d'algorithmes de planification de chemins classiques (A*, RRT*-Connect) et d'approches par optimisation. Un contrôleur simple PID et un contrôleur *MPC* ont aussi été implémentés, permettant la mise en place de stratégies d'évitement d'obstacles statiques et dynamiques. Un simulateur Python basique a été créé pour faciliter le développement des divers

algorithmes et pour permettre de les tester. Les résultats expérimentaux, obtenus aussi bien sur le simulateur Python que sur Gazebo, montrent que les méthodes classiques se distinguent par leur rapidité et leur efficacité dans des environnements simples, tandis que les approches basées sur l'optimisation, bien que plus coûteuses en calcul, offrent une meilleure intégration des contraintes et un potentiel supérieur dans des contextes dynamiques complexes. Ces contributions ont été intégrées dans un package Python modulaire et un package ROS2, facilitant ainsi leur utilisation et leur extension future.

Ce travail présente toutefois des limites importantes. Les méthodes de génération de trajectoires par optimisation souffrent encore de temps de calcul élevés, ce qui limite leur applicabilité en temps réel. De plus, bien que les tests sur simulateur aient été partiellement concluants, les validations sur robot réel n'ont pas encore pu être menées. Enfin, le suivi des obstacles reste basé sur une représentation abstraite et simplifiée utilisant des obstacles virtuels.

Plusieurs pistes d'amélioration sont envisageables. À court terme, l'optimisation du calcul symbolique, l'utilisation de techniques de *warm-start* et la standardisation des dépendances devraient permettre une amélioration significative des performances de toutes les stratégies développées. À moyen terme, les questions de projection de la référence en dehors des zones interdites et de gestion des obstacles approfondie sont des axes de travail majeurs à explorer. Enfin, des essais sur le robot Tiago doivent être envisagés pour valider expérimentalement les approches développées.

Pour conclure, ce travail constitue une base solide pour la génération de trajectoires et la commande de robots mobiles en environnement dynamique et il pourrait être groupé à d'autres travaux réalisés au sein de l'équipe pour déboucher sur une publication, moyennant l'intégration des pistes d'amélioration évoquées. Enfin, d'un point de vue personnel, ce stage a confirmé mon intérêt pour la commande avancée de manipulateurs mobiles. Je suis donc très enthousiaste de pouvoir continuer à explorer ce domaine dans le cadre de la thèse que je vais débuter à la rentrée prochaine.

Acronymes

AMCL Localisation adaptative de Monte Carlo.
BIATSS personnels des bibliothèques, ingénieurs, administratifs, techniques et sociaux et de santé.
CNES Centre National d'Études Spatiales.
CNRS Centre National de la Recherche Scientifique.
DMC Dynamic Matrix Control.
EKF Extended Kalman Filter.
ENSTA École Nationale Supérieure de Techniques Avancées.
ITA ingénieurs, techniciens et administratifs.
LAAS Laboratoire d'Analyse et d'Architecture des Systèmes.
MPC Model Predictive Control.
OMPL Open Motion Planning Library.
RAP Robotique, Action, Perception.
RIS Robotique et Interactions.

Table des Figures

Liste des Tableaux

Annexes

Annexe A : Fiche d'appréciation

FICHE D'APPRECIATION DE STAGE

A renseigner et à viser par le tuteur entreprise puis faire retour sous aurion rubrique « mise à jour de PFE »

Organisme	LAAS-CNRS
Dates du stage	17/03/2025 - 29/08/2025
NOM, Prénom du stagiaire	RANGARADJOU Harendra

	Cocher les cases appropriées					
	F (échec)	E (insuffisant)	D (passable)	C (assez bien à bien)	B (bien à très bien)	A (remarquable)
Critères d'intégration – Savoir être						
Adaptabilité					X	
Disponibilité					X	
Culture de l'entreprise					X	
Puissance de travail					X	
Qualité d'expression					X	
Conduite du projet						
Identification des tâches					X	
Organisation/répartition des tâches dans le temps				X		
Respect des délais des livrables demandés				X		
Force de proposition						X
Éventuellement : travail en équipe					X	
Rapport de stage						
Forme (présentation, style…)					X	
Fond (exactitude)					X	
Exploitabilité par l'organisme					X	

Appréciation de la formation ENSTA Bretagne						
Les compétences scientifiques et techniques répondent à mes attendus					X	
Les compétences méthodologiques répondent à mes attendus					X	
Sur quels sujets a-t-il fallu former le stagiaire avant qu'il ne soit autonome ?	Presque rien, quelques discussions sur des aspects de commande					
Quelles seraient les compétences ou les contenus de formation à renforcer ?	Rien à Signaler					

Appréciation générale

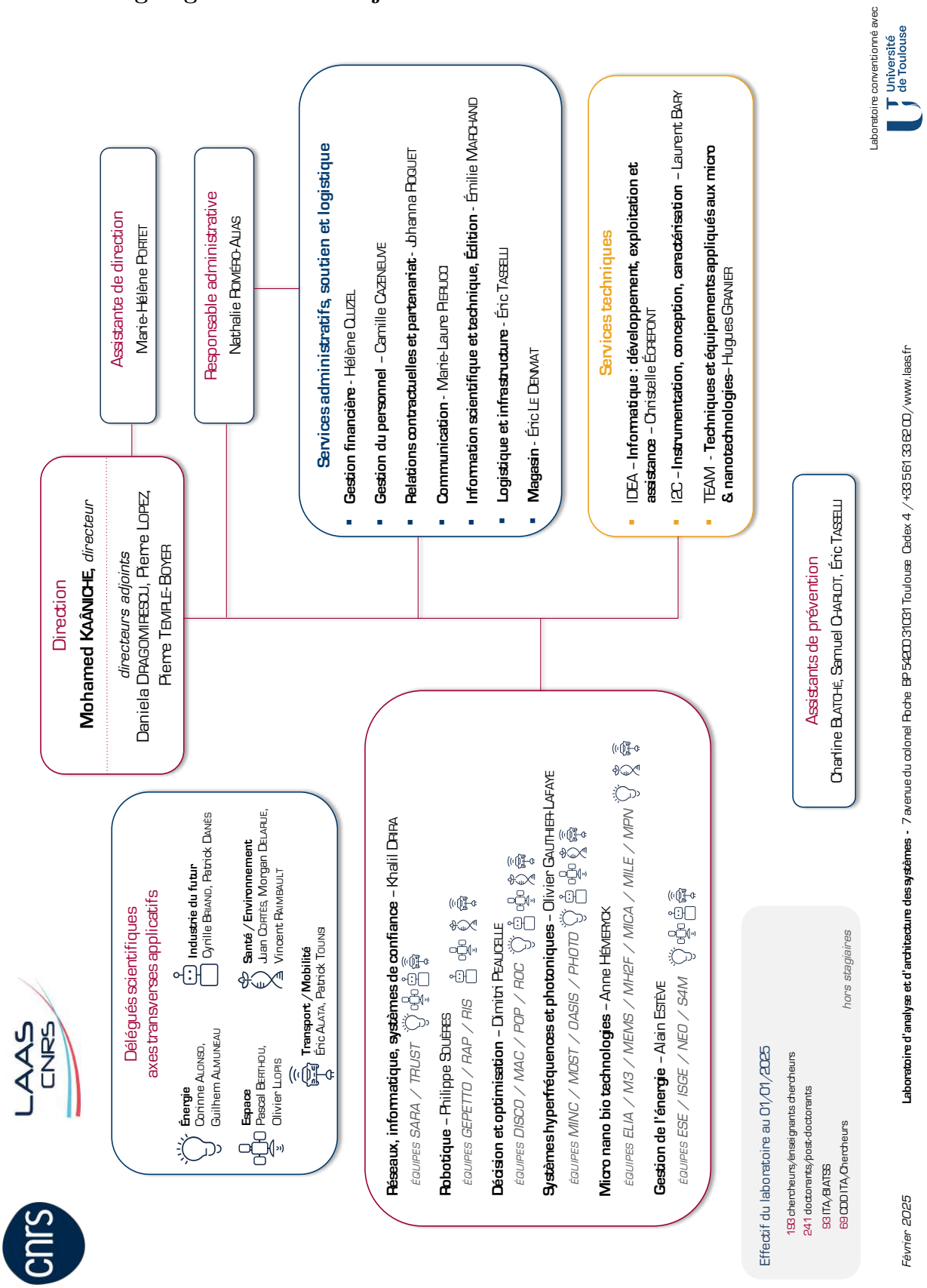
Harendra a travaillé de manière autonome et a démontré une bonne capacité à réaliser les tâches qui lui ont été confiées. Il a su s'approprier le sujet, proposer des solutions pertinentes et analyser les difficultés rencontrées. Ses résultats constituent une base intéressante, offrant des pistes pour identifier les verrous et problématiques à traiter dans les travaux futurs.

Si vous disposiez d'un poste correspondant au profil du stagiaire, souhaiteriez-vous lui proposer ? ☒ **OUI** ☐ **NON**

NOM, Prénom du tuteur entreprise : MUJICA, Martin **Date :** 12/08/2025

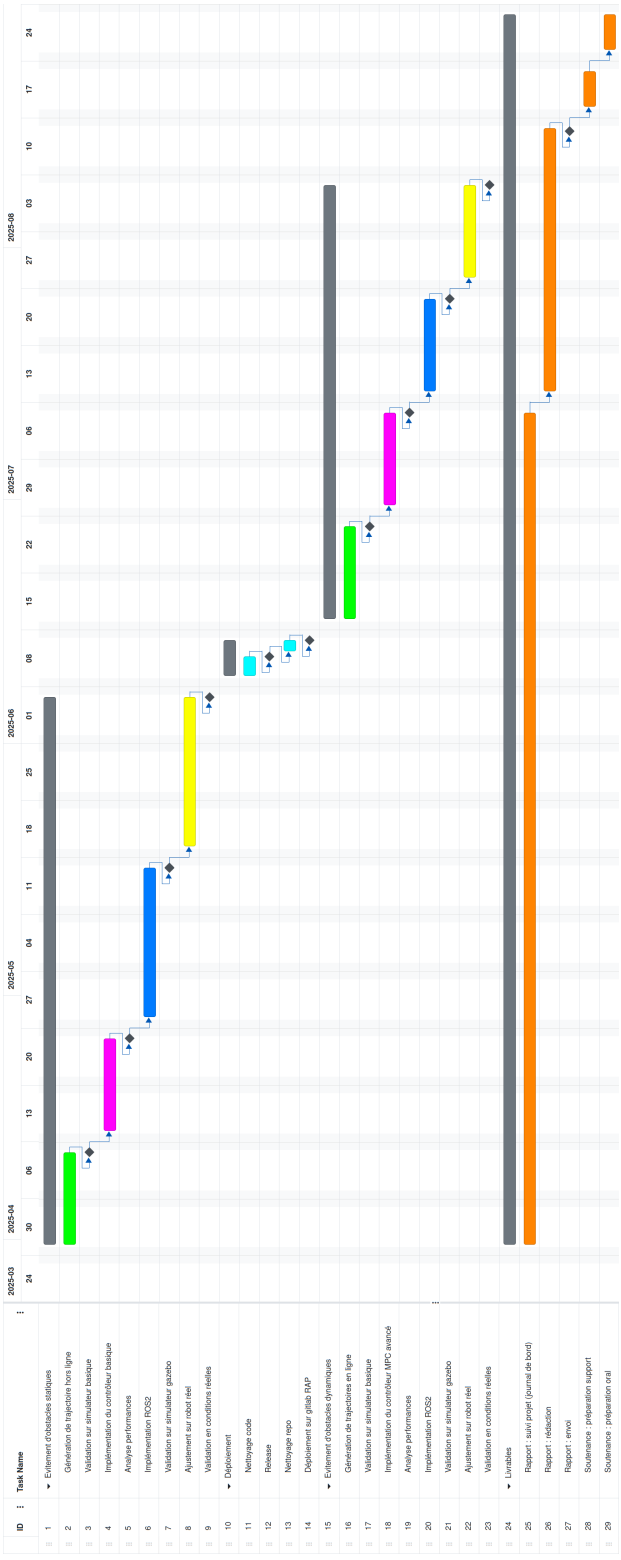
Fonction : Maître de Conférences **Signature :**

Annexe B : Organigramme *LAAS* janvier 2025

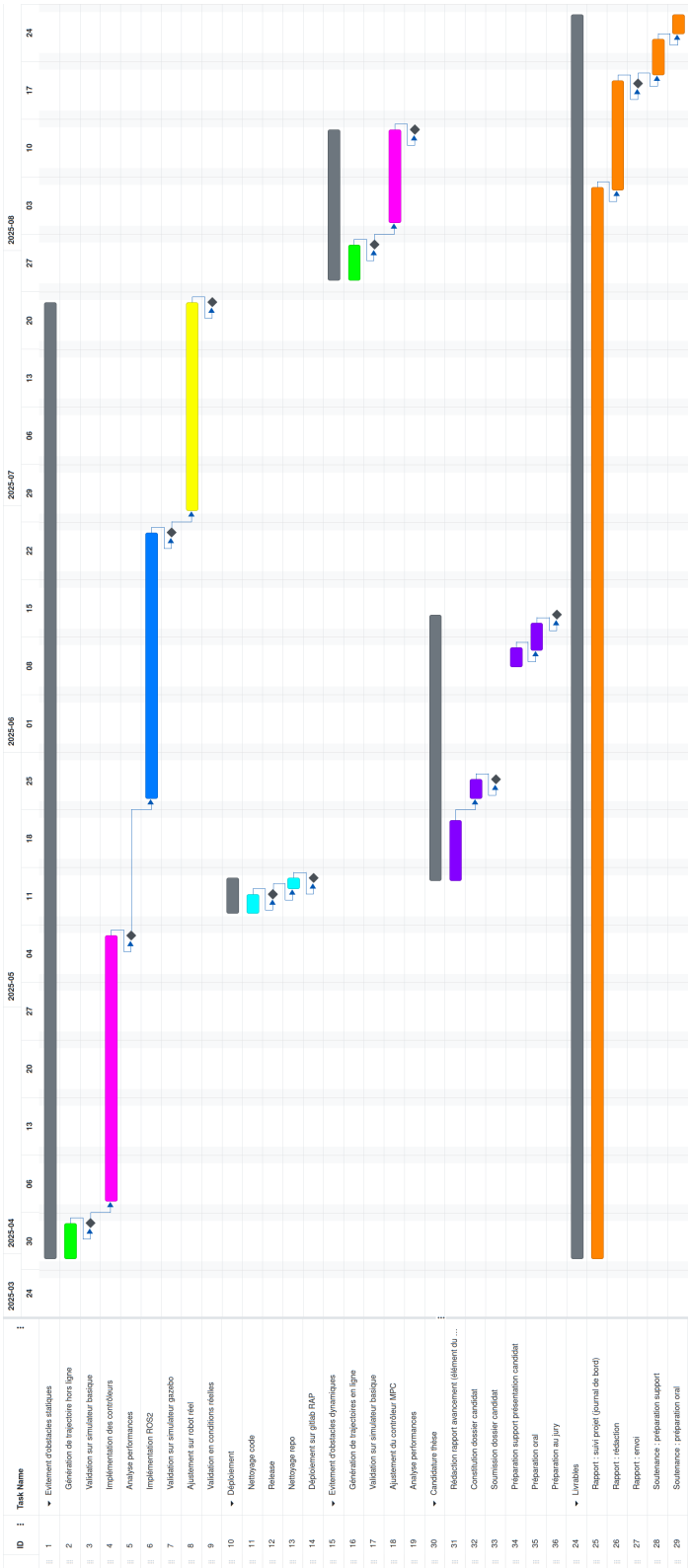


Annexe C : Diagrammes de Gantt

Annexe C-1 : Organisation prévue



Annexe C-2 : Déroulement réel des activités



Annexe D : Algorithmes et pseudo-code

Annexe D-1 : Filtrage des chemins obtenus par optimisation

Algorithme 1 Filtrage de chemin

procedure FILTERPATH($path$)
    $resolution \leftarrow 0.2 \cdot v_{max} \cdot \Delta t$
    $heading\_variation\_threshold \leftarrow 3$ ▷ degrés
    $max\_artifact\_size \leftarrow 50$
    $min\_artifact\_size \leftarrow 5$
    $filtered\_path \leftarrow [path[:, 0]]$
    $nb\_samples\_raw \leftarrow$ nombre de points dans $path$
    $i \leftarrow 0$
    while $i < (nb\_samples\_raw - max\_artifact\_size)$ **do**
        $loop\_detected \leftarrow$ **False**
        for $j = min\_artifact\_size$ **to** $max\_artifact\_size$ **do**
            $x_{curr} \leftarrow path[:2, i]$
            $x_{next} \leftarrow path[:2, i + j]$
            if $\|x_{next} - x_{curr}\| < resolution$ **and** $\text{std}(path[2, i : i + j]) < heading\_variation\_threshold$ **then**
                $loop\_exit\_index \leftarrow j$
                $loop\_detected \leftarrow$ **True**
            end if
        end for
        if $loop\_detected$ **and** $loop\_exit\_index \geq min\_artifact\_size$ **then**
            Ignorer les $loop\_exit\_index$ prochains points
            $i \leftarrow i + loop\_exit\_index$
        else
            $i \leftarrow i + 1$
        end if
        Ajouter $path[:, i]$ à $filtered\_path$
    end while
    $filtered\_path \leftarrow$ concaténation de $filtered\_path$ et de la fin de $path$ (longueur $max\_artifact\_size - 1$)
    return $filtered\_path$
end procedure

Annexe D-2 : Association entre points du chemin et points de passage

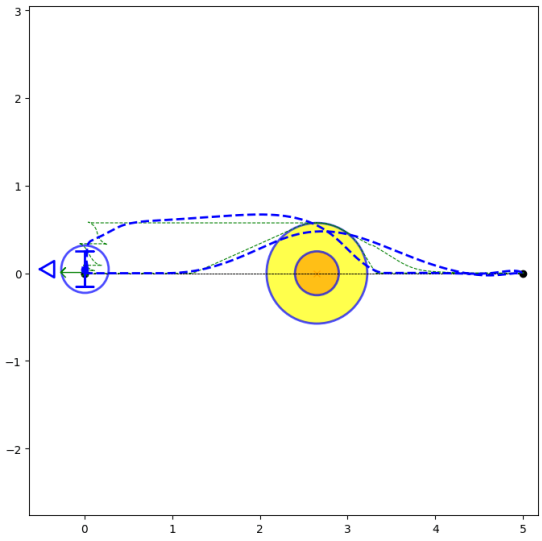
Algorithme 2 Association de points

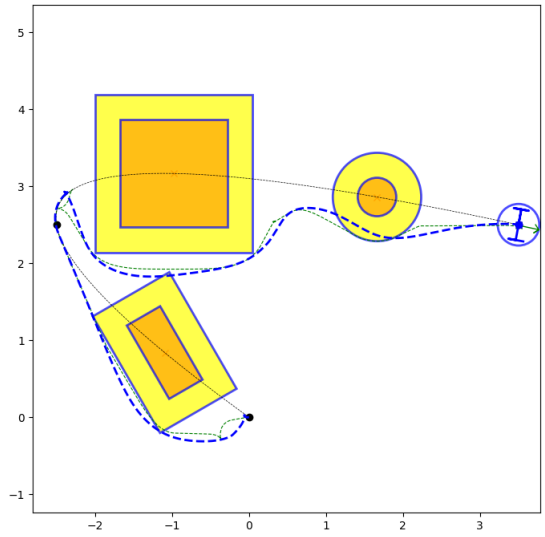
procedure UPDATEWAYPOINTINDICES($waypoints, path$)
  waypoint_indices $\leftarrow [0]$ ▷ Le premier waypoint est toujours le début du chemin
  for $k, wp \in$ Enumerate($waypoints^T[1:-1]$) **do**
    if $\|wp - waypoints[:, k+2]\| <$ rob.size $\cdot\, 10^{-6}$ **then**
      return ▷ "Waypoint $k+2$ too close to waypoint $k+3$"
    end if
    path_sample_sizes $\leftarrow \sqrt{\Delta x^2 + \Delta y^2}$ ▷ Variations entre points consécutifs du chemin
    search_radius $\leftarrow$ max(path_sample_sizes)
    distances $\leftarrow \|path[:2] - wp\|_2$
    indexed_distances $\leftarrow \{(i, dist)\}$ triés par *distance*
    sorted_indexed_distances $\leftarrow$ indexed_distances$[\#\{dist \leq$ search_radius$\}]$
    Trier sorted_indexed_distances par *index*
    sorted_indices $\leftarrow$ sorted_indexed_distances.index
    closest_index $\leftarrow$ min{sorted_indices$[j]$ | sorted_indices$[j]$ − waypoint_indices$[-1] > 0$}
    Ajouter closest_index à waypoint_indices
  end for
  Ajouter (nb_samples − 1) à waypoint_indices ▷ Le dernier waypoint est la fin du chemin
end procedure

Annexe E : Traces obtenues pour l'évitement d'obstacles statiques (axes gradués en mètres)

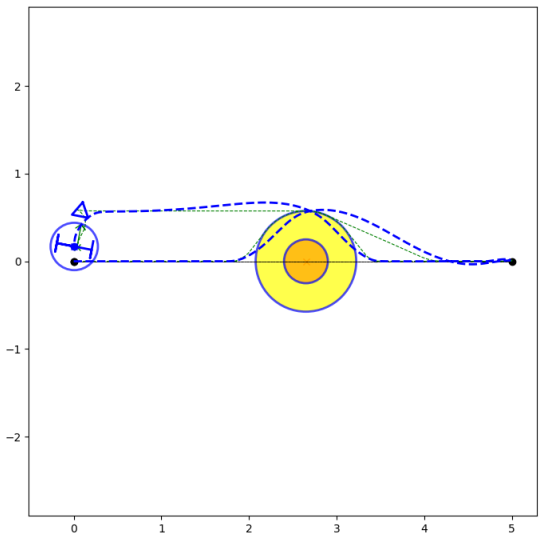
Annexe E-1 : Traces obtenues sur le simulateur Python

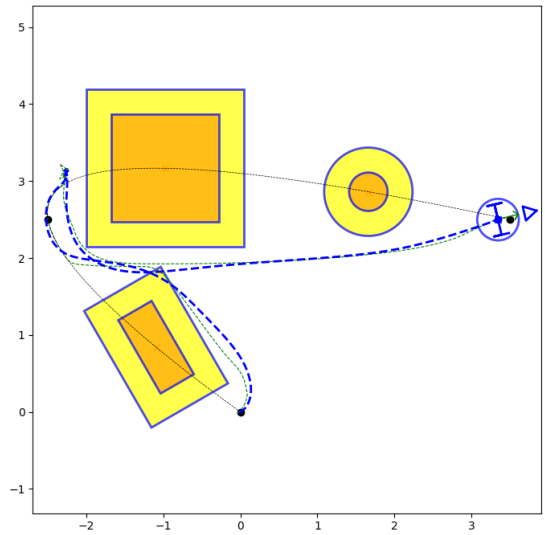
Traces obtenues par optimisations adjacentes :





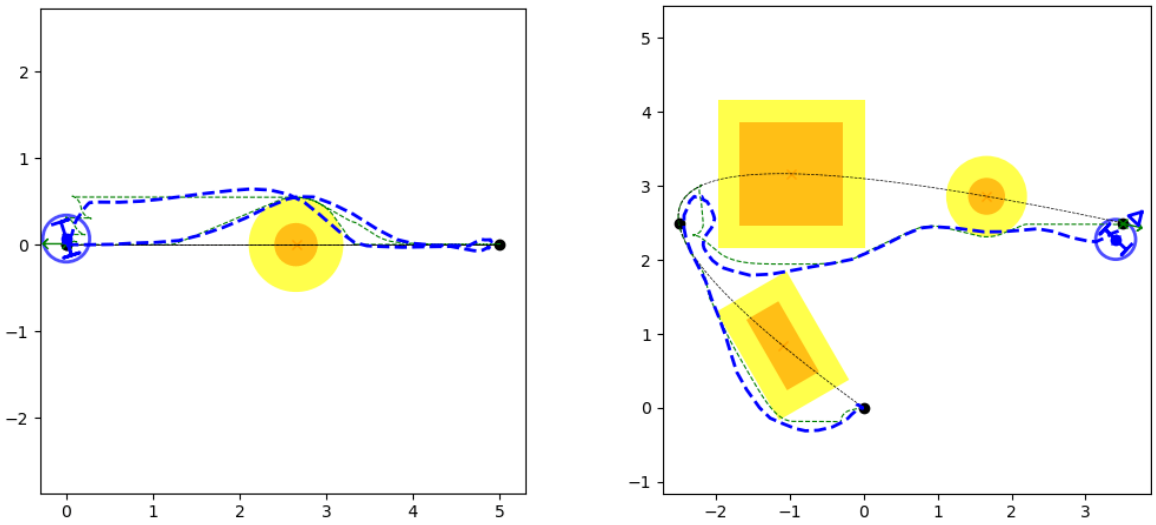
Traces obtenues par optimisation en amont :



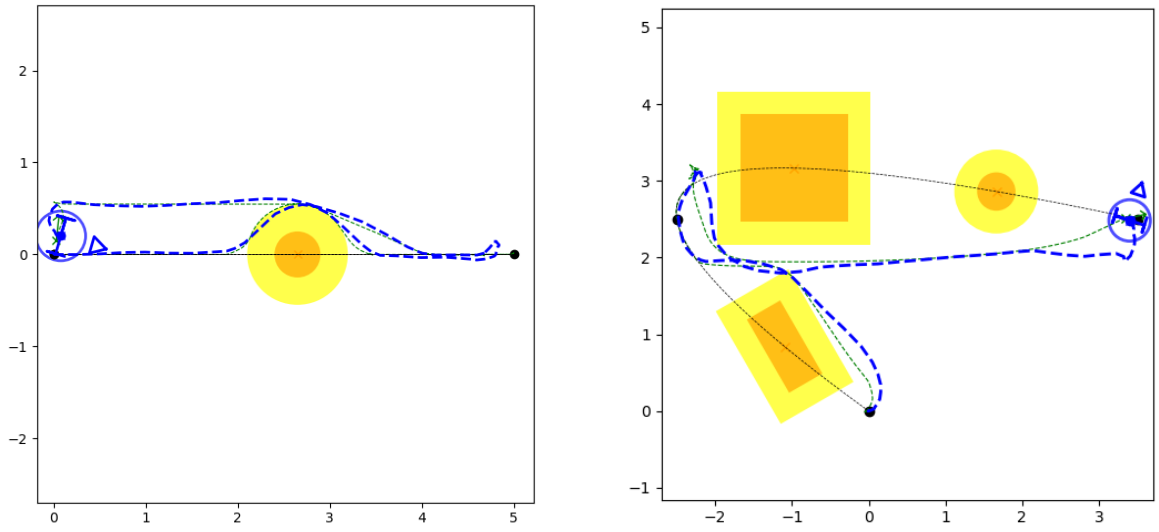


Annexe E-2 : Traces obtenues sur le simulateur Gazebo

Traces obtenues par optimisations adjacentes :

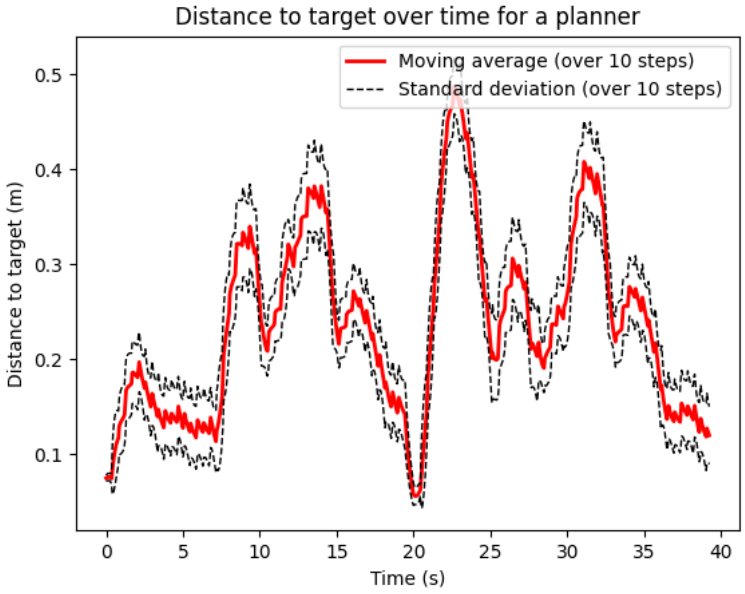


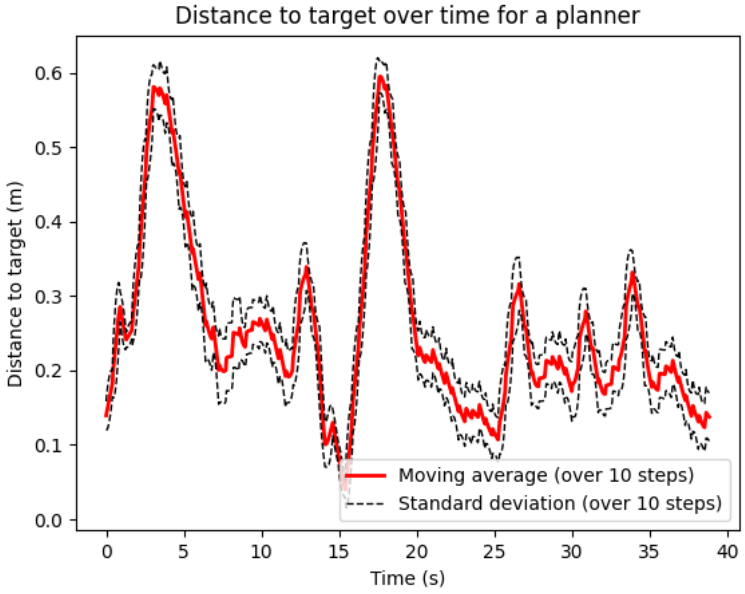
Traces obtenues par optimisation en amont :



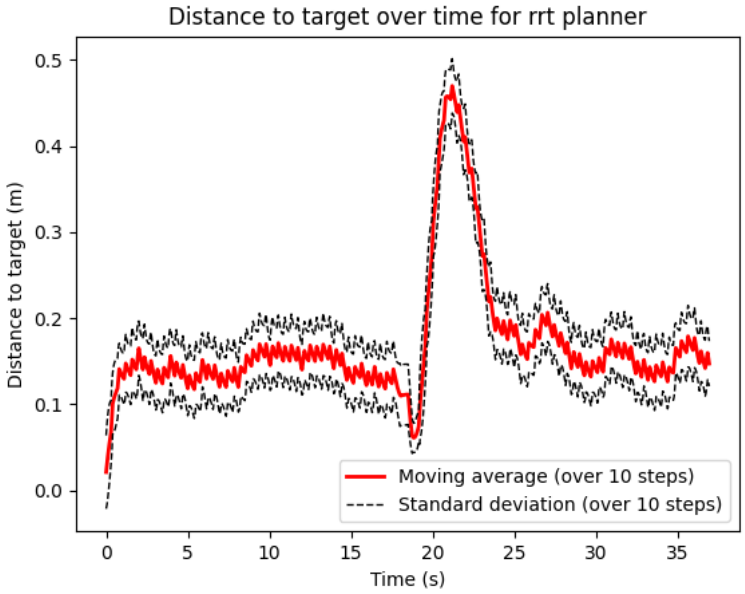
Annexe F : Graphes des distances à la référence

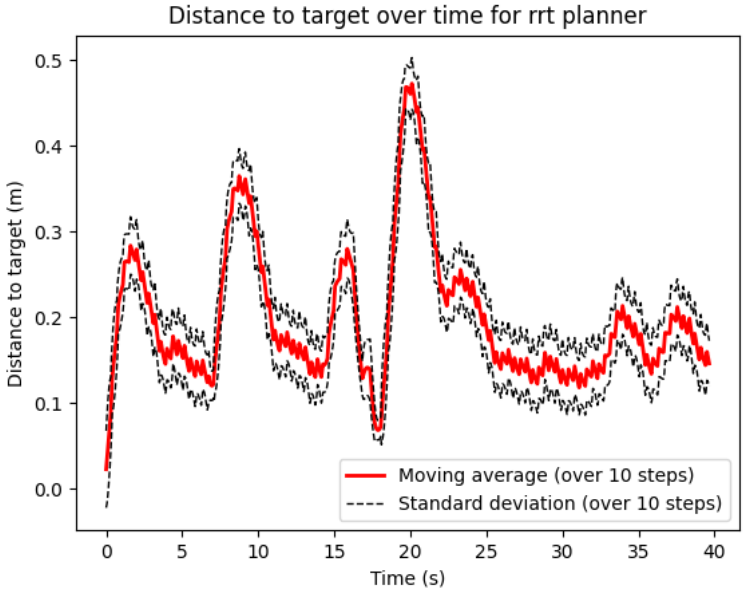
Annexe F-1 : Graphes obtenus pour une trajectoire de référence générée par l'algorithme A* modifié



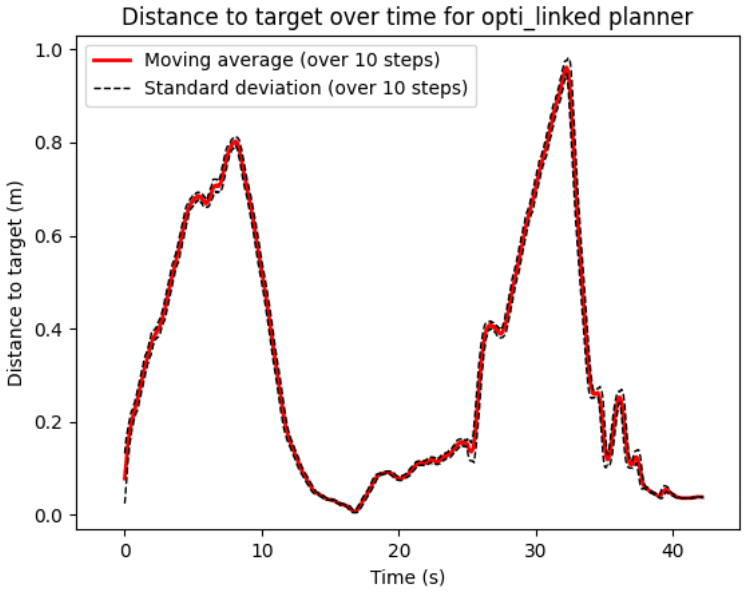


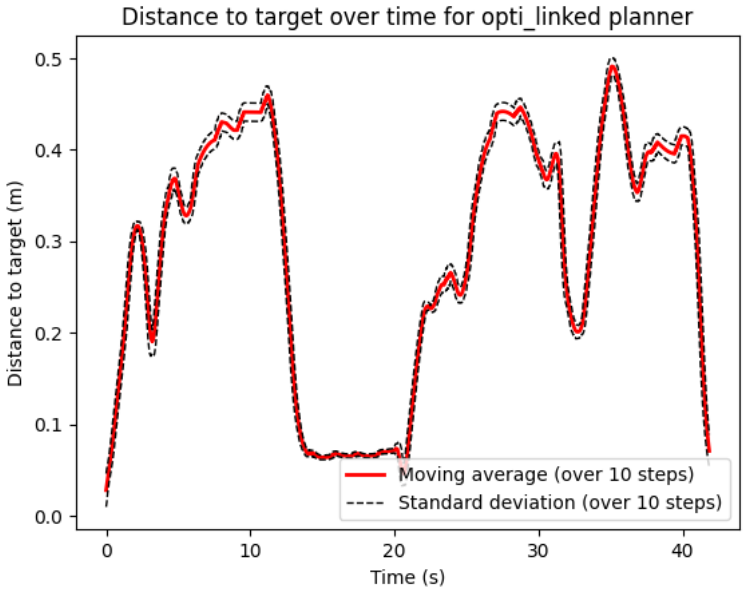
Annexe F-2 : Graphes obtenus pour une trajectoire de référence générée par l'algorithme RRT*-connect modifié



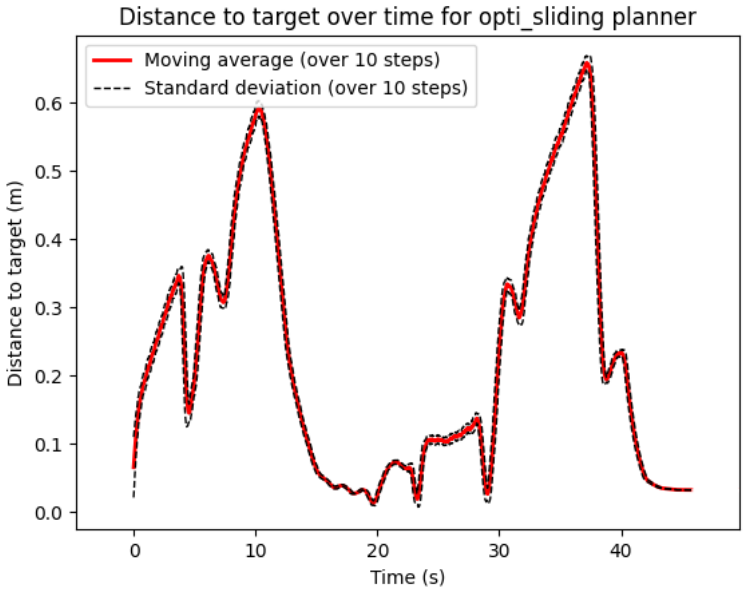


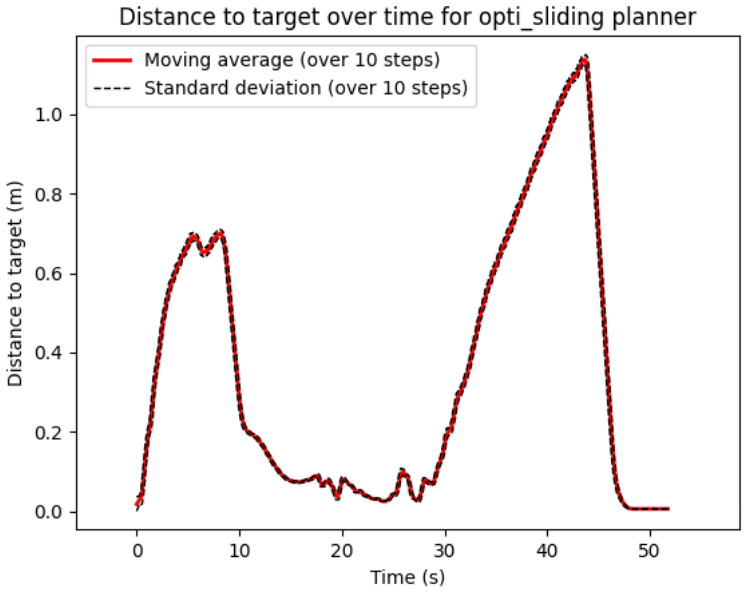
Annexe F-3 : Graphes obtenus pour une trajectoire de référence générée par la méthode des optimisations adjacentes





Annexe F-4 : Graphes obtenus pour une trajectoire de référence générée par la méthode de l'optimisation en amont





Références

[1] Martin Klaučo et Michal Kvasnica. *MPC-Based Reference Governors: Theory and Case Studies*. en. Advances in Industrial Control. Cham : Springer International Publishing, 2019, p. 1-5. ISBN : 978-3-030-17404-0 978-3-030-17405-7. DOI : 10.1007/978-3-030-17405-7. URL : http://link.springer.com/10.1007/978-3-030-17405-7 (visité le 24/03/2025).

[2] Jie Ji et al. « Path Planning and Tracking for Vehicle Collision Avoidance Based on Model Predictive Control With Multiconstraints ». In : *IEEE Transactions on Vehicular Technology* 66.2 (fév. 2017), p. 952-964. ISSN : 0018-9545, 1939-9359. DOI : 10.1109/TVT.2016.2555853. URL : http://ieeexplore.ieee.org/document/7458179/ (visité le 14/08/2025).

[3] Neel P. Bhatt, Amir Khajepour et Ehsan Hashemi. « MPC-PF: Socially and Spatially Aware Object Trajectory Prediction for Autonomous Driving Systems Using Potential Fields ». In : *IEEE Transactions on Intelligent Transportation Systems* 24.5 (mai 2023), p. 5351-5361. ISSN : 1558-0016. DOI : 10.1109/TITS.2023.3243004. URL : https://ieeexplore.ieee.org/abstract/document/10046400 (visité le 14/08/2025).

[4] Chang Liu et al. « Path planning for autonomous vehicles using model predictive control ». In : *2017 IEEE Intelligent Vehicles Symposium (IV)*. Los Angeles, CA, USA : IEEE, juin 2017, p. 174-179. ISBN : 978-1-5090-4804-5. DOI : 10.1109/IVS.2017.7995716. URL : http://ieeexplore.ieee.org/document/7995716/ (visité le 14/08/2025).

[5] Carlos E. Luis et Angela P. Schoellig. « Trajectory Generation for Multiagent Point-To-Point Transitions via Distributed Model Predictive Control ». In : *IEEE Robotics and Automation Letters* 4.2 (avr. 2019), p. 375-382. ISSN : 2377-3766. DOI : 10.1109/LRA.2018.2890572. URL : https://ieeexplore.ieee.org/abstract/document/8598938 (visité le 15/08/2025).

[6] M. Mahdi Ghazaei Ardakani et al. « Model Predictive Control for Real-Time Point-to-Point Trajectory Generation ». In : *IEEE Transactions on Automation Science and Engineering* 16.2 (avr. 2019). Conference Name: IEEE Transactions on Automation Science and Engineering, p. 972-983. ISSN : 1558-3783. DOI : 10.1109/TASE.2018.2882764. URL : https://ieeexplore.ieee.org/document/8586948/?arnumber=8586948 (visité le 24/03/2025).

[7] Nigora Gafur et al. « Dynamic path planning and reactive scheduling for a robotic manipulator using nonlinear model predictive control ». In : *2022 30th Mediterranean Conference on Control and Automation (MED)*. ISSN: 2473-3504. Juin 2022, p. 604-611. DOI : 10.1109/MED54222.2022.9837147. URL : https://ieeexplore.ieee.org/abstract/document/9837147 (visité le 15/08/2025).

[8] Chaocheng Li et al. « A model based path planning algorithm for self-driving cars in dynamic environment ». In : *2015 Chinese Automation Congress (CAC)*. Wuhan, China : IEEE, nov. 2015, p. 1123-1128. ISBN : 978-1-4673-7189-6. DOI : 10.1109/CAC.2015.7382666. URL : http://ieeexplore.ieee.org/document/7382666/ (visité le 15/08/2025).

[9] Alan Bundy et Lincoln Wallen. « Breadth-First Search ». en. In : *Catalogue of Artificial Intelligence Tools*. Sous la dir. d'Alan Bundy et Lincoln Wallen. Berlin, Heidelberg : Springer, 1984, p. 13-13. ISBN : 978-3-642-96868-6. DOI : 10.1007/978-3-642-96868-6_25. URL : https://doi.org/10.1007/978-3-642-96868-6_25 (visité le 28/05/2025).

[10] E. W. Dijkstra. « A note on two problems in connexion with graphs ». en. In : *Numerische Mathematik* 1.1 (déc. 1959), p. 269-271. ISSN : 0029-599X, 0945-3245. DOI : 10.1007/BF01386390. URL : http://link.springer.com/10.1007/BF01386390 (visité le 28/05/2025).

[11] Peter E. Hart, Nils J. Nilsson et Bertram Raphael. « A Formal Basis for the Heuristic Determination of Minimum Cost Paths ». In : *IEEE Transactions on Systems Science and Cybernetics* 4.2 (juill. 1968), p. 100-107. ISSN : 2168-2887. DOI : 10.1109/TSSC.1968.300136. URL : https://ieeexplore.ieee.org/abstract/document/4082128 (visité le 28/05/2025).

[12] Sertac Karaman et Emilio Frazzoli. « Sampling-based algorithms for optimal motion planning ». en. In : *The International Journal of Robotics Research* 30.7 (juin 2011). Publisher: SAGE Publications Ltd STM, p. 846-894. ISSN : 0278-3649. DOI : 10.1177/0278364911406761. URL : https://doi.org/10.1177/0278364911406761 (visité le 18/01/2025).

[13] Jonathan D. Gammell, Siddhartha S. Srinivasa et Timothy D. Barfoot. « Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic ». In : *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. ISSN: 2153-0866. Sept. 2014, p. 2997-3004. DOI : 10.1109/IROS.2014.6942976. URL : https://ieeexplore.ieee.org/abstract/document/6942976?casa_token=ZXXnQwykfG8AAAAA:Ner5_gY-jRUj0oyMacI0mCeCJs_4zs51-x_wG8Rnx6lraYmHvmoSHQPg0Uftn-Z2nD2kvgc6qA (visité le 18/01/2025).

[14] Sebastian Klemm et al. « RRT*-Connect: Faster, asymptotically optimal motion planning ». In : *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. Zhuhai, China : IEEE, déc. 2015, p. 1670-1677. ISBN : 978-1-4673-9675-2. DOI : 10.1109/ROBIO.2015.7419012. URL : https://ieeexplore.ieee.org/document/7419012/ (visité le 18/01/2025).

[15] Ángel Llamazares, Eduardo J. Molinos et Manuel Ocaña. « Detection and Tracking of Moving Obstacles (DATMO): A Review ». en. In : *Robotica* 38.5 (mai 2020), p. 761-774. ISSN : 0263-5747, 1469-8668. DOI : 10.1017/S0263574719001024. URL : https://www.cambridge.org/core/journals/robotica/article/abs/detection-and-tracking-of-moving-obstacles-datmo-a-review/BB94A95B06491BE09227A8BE7EDB7777 (visité le 23/06/2025).

[16] Luis Montesano, Javier Minguez et Luis Montano. « Modeling the Static and the Dynamic Parts of the Environment to Improve Sensor-based Navigation ». In : mai 2005, p. 4556-4562. DOI : 10.1109/ROBOT.2005.1570822.

[17] Mukhtar Sani, Bogdan Robu et Ahmad Hably. « Dynamic Obstacles Avoidance Using Nonlinear Model Predictive Control ». In : *IECON 2021 – 47th Annual Conference of the IEEE Industrial Electronics Society*. ISSN: 2577-1647. Oct. 2021, p. 1-6. DOI : 10.1109/IECON48115.2021.9589658. URL : https://ieeexplore.ieee.org/abstract/document/9589658 (visité le 28/07/2025).

[18] Joel A. E. Andersson et al. « CasADi – A software framework for nonlinear optimization and optimal control ». In : *Mathematical Programming Computation* (2018).

[19] Andreas Wächter et Lorenz T. Biegler. « On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming ». In : *Mathematical Programming* 106.1 (1er mars 2006), p. 25-57. ISSN : 1436-4646. DOI : 10.1007/s10107-004-0559-y. URL : https://doi.org/10.1007/s10107-004-0559-y (visité le 27/05/2025).

[20] Ioan A. Şucan, Mark Moll et Lydia E. Kavraki. « The Open Motion Planning Library ». In : *IEEE Robotics & Automation Magazine* 19.4 (déc. 2012). https://ompl.kavrakilab.org, p. 72-82. DOI : 10.1109/MRA.2012.2205651.

[21] Steven Macenski et al. « Robot Operating System 2: Design, architecture, and uses in the wild ». In : *Science Robotics* 7.66 (2022), eabm6074. doi : 10.1126/scirobotics.abm6074. url : https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

[22] Robin Verschueren et al. « acados – a modular open-source framework for fast embedded optimal control ». In : *Mathematical Programming Computation* (2021).