Identification, planification et suivi de trajectoire avec un robot.

Lucas ARIES
École d'ingénieur informatique
TELECOM Nancy
Villers-lès-Nancy, France
Email: lucas.aries@telecomnancy.eu

Aurélie DEMURE
École d'ingénieur informatique
TELECOM Nancy
Villers-lès-Nancy, France
Email: aurelie.demure@telecomnancy.eu

Julie ZHEN
École d'ingénieur informatique
TELECOM Nancy
Villers-lès-Nancy, France
Email: julie.zhen@telecomnancy.eu

Amine BOUMAZA
Laboratoire lorrain de recherche
en informatique et ses applications
LORIA, Campus scientifique BP 239
Vandoeuvre-lès-Nancy Cedex, France
Email: amine.boumaza@loria.fr

***Abstract*—L'objectif de cette étude est de développer un système de contrôle autonome pour un robot, en utilisant un programme informatique conjointement avec des données provenant d'une caméra. Ce système vise à permettre au robot de naviguer de manière autonome d'un point A à un point B en utilisant diverses solutions techniques.**
Mots-clés: Robot Thymio, Robotique, Traitement d'image, Planification de trajectoire

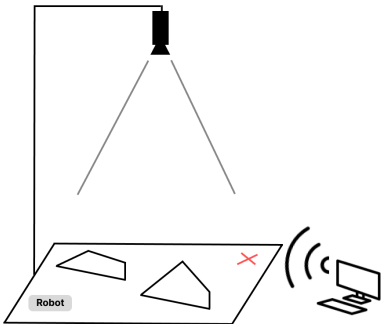

Fig. 1. Représentation de l'arène

I. Introduction

Dans ce projet, nous souhaitons réaliser un prototype de plate-forme pour l'étude de la coordination robotique. Il s'agira d'une arène qui permettrait de de faire évoluer un robot Thymio2 dans des environnements modulables. D'un point de vue technique, l'arène sera dotée d'un système de captation par caméras pour localiser les objets et le robot. Un ordinateur se chargera de la programmation et la communication avec le robot.

A. Les problématiques

1) Communication avec le robot: Dans ce contexte, notre objectif est de développer un système de contrôle pour un robot nécessitant une communication permanente avec une source externe, plutôt que de téléverser un programme directement sur le robot pour une autonomie complète. Ce système doit permettre au robot de naviguer de manière autonome d'un point A à un point B en utilisant des informations provenant d'une caméra et en employant diverses solutions techniques.

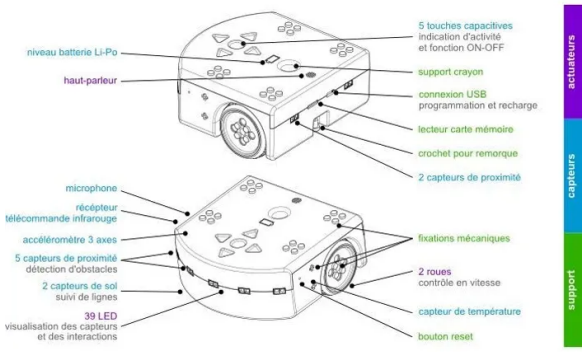

Fig. 2. Représentation d'un robot Thymio2

2) Détection des obstacles et du robot: Il faut explorer la problématique de la détection et de l'identification d'objets dans une scène à l'aide d'une caméra positionnée en hauteur pour obtenir une vue en deux dimensions. Plus spécifiquement le défi de repérer un robot ainsi que divers obstacles dans un environnement donné

3) Planification de trajectoire: La problématique soulevée ici est la nécessité de planifier la trajectoire d'un robot en utilisant les données fournies par une caméra. Cette planification requiert la conversion de ces données dans un format approprié pour les algorithmes de planification classiques.

4) Suivi de trajectoire: La problématique abordée concerne l'optimisation de la navigation du robot afin qu'il suive la trajectoire planifiée de manière fluide, sans heurts ni à-coups. L'enjeu réside dans la réalisation d'un contrôle précis et

efficace du robot pour garantir une expérience de déplacement homogène et sans accrocs.

II. Communication avec le robot

Dans notre contexte actuel, nous identifions la communication entre le robot et son environnement comme un élément crucial pour le développement d'une solution efficace. Cependant, nous devons prendre en compte des contraintes physiques et logicielles.

La contrainte physique principale est la portée entre le robot et son ordinateur, ainsi que l'utilisation d'une caméra pour la perception de l'environnement. L'utilisation d'un câble pour la communication entre le robot et l'ordinateur est donc exclue, car cette solution limiterait considérablement la mobilité du robot et ajouterait des informations inutiles sur le flux vidéo.

Pour répondre à cette contrainte, nous utilisons une connexion sans fil pour la communication entre le robot et l'ordinateur. Le robot Thymio offre déjà une solution de communication sans fil, ce qui facilite la mise en œuvre de notre solution. Nous utilisons la technologie Bluetooth pour assurer une communication fiable et efficace entre le robot et l'ordinateur.

Dans le cadre de notre projet, nous devons assurer une communication permanente entre le robot et l'ordinateur afin de pouvoir adapter la trajectoire du robot en temps réel en fonction des changements du terrain. En effet, le robot doit être notifié d'un nouveau chemin calculé par l'ordinateur et être en mesure de changer de trajectoire en conséquence.

C'est pourquoi nous ne pouvons pas téléverser un programme au robot et le laisser se déplacer de manière autonome, car cela ne répond pas aux exigences de notre projet. Pour répondre à cette problématique, nous utilisons la librairie ThymioDirect qui permet de communiquer en permanence avec le robot et de lui donner des instructions depuis l'ordinateur en temps réel et sans délai. Cette bibliothèque nous permet de contrôler le robot à distance et de mettre à jour sa trajectoire en fonction des données envoyées par l'ordinateur.

Dans un souci de facilité de développement et de compréhension, nous avons créé une classe pour le robot qui sert d'adaptateur à la bibliothèque. Cela facilite considérablement l'utilisation du robot dans le projet en fournissant une interface simple (il n'est pas nécessaire de configurer le robot et de se reconnecter à chaque fois que l'on souhaite lui donner de nouvelles instructions).

III. Détection des obstacles et du robot

Il est essentiel de détecter les obstacles et les robots présents sur le terrain afin de pouvoir effectuer des traitements sur ces derniers. Plusieurs solutions sont disponibles, et nous avons sélectionné celles qui nous semblaient les plus pertinentes.

A. Détection du robot

Pour identifier le robot, nous allons utiliser des codes ArUco que nous placerons au centre de celui-ci, de manière à ce qu'ils

Fig. 3. Exemple de code ArUco

soient facilement visibles depuis la caméra qui sera placée en hauteur.

Ils ressemblent à des QR codes, mais ils contiennent beaucoup moins d'informations et ont donc moins de détails. Cela permet d'avoir une résolution plus faible sur la caméra, car l'information est plus facilement lisible de loin et de mauvaise qualité. Cela nous permet de positionner la caméra en retrait et d'obtenir ainsi un champ de vision beaucoup plus large qu'avec une autre solution.

En effet, la résolution nécessaire pour récupérer l'information est beaucoup moins importante. Contrairement au QR code qui possède des informations redondantes pour ne pas avoir d'orientation, le code ArUco a une orientation définie, ce qui nous sera utile par la suite.

1) Détection ArUco: Pour détecter les codes ArUco, nous allons utiliser la bibliothèque OpenCV et son module ArUco.

Fig. 4. Exemple détection de code ArUco

Une fois le code détecté, il nous donne une orientation, ce qui nous permet de déduire l'orientation du robot auquel le code est rattaché.

Premièrement il faut récupérer les axes qui sont pertinentes

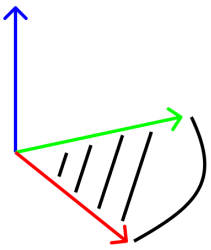

Fig. 5. Axes 3

Nous avons utilisé la bibliothèque OpenCV pour obtenir des vecteurs de rotation. Cependant, pour faires des calculs nous devons convertir ces vecteurs en matrices de rotation. Pour ce faire, nous avons utilisé la méthode de Rodrigues, qui permet de dériver une matrice de rotation à partir d'un vecteur de rotation donné.

La méthode de Rodrigues est basée sur la formule suivante :

$$R = I + sin(\theta)S + (1 - cos(\theta))S^2$$

où R est la matrice de rotation, I est la matrice identité, est la norme du vecteur de rotation et S est la matrice antisymétrique associée au vecteur de rotation.

Une fois la matrice de rotation obtenue, nous avons pu récupérer les angles d'Euler correspondants en utilisant la fonction cv2.decomposeProjectionMatrix de OpenCV. Les angles d'Euler représentent les rotations autour des trois axes de l'espace et peuvent être utilisés pour analyser les angles

Dans notre cas, nous nous sommes intéressés à l'angle de rotation autour de l'axe vertical, qui correspond à la rotation de l'objet autour de son axe de symétrie vertical.

Pour adapter les valeurs de l'angle à notre problème spécifique, il est nécessaire de les recentrer.

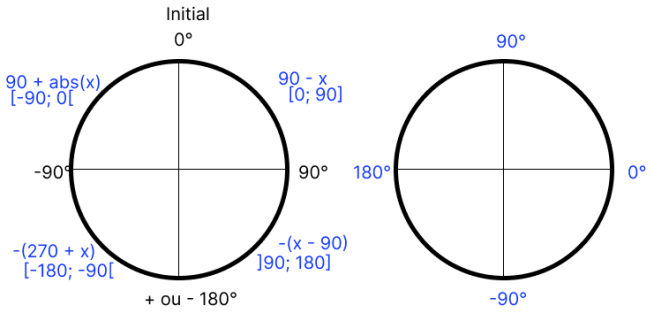

Fig. 6. Transformation de la valeur des angles

IV. Planification de trajectoire

A. Les algorithmes de planification de trajectoire

Afin que notre robot puisse aller d'un point A à un point B, nous avons décidé d'utiliser un algorithme de recherche de chemin basé dans un repère. Voici un état de l'art de tous les algorithmes existants.

1) L'algorithme de parcours en largeur: Cet algorithme permet le parcours récursif d'un graphe ou d'un arbre à partir d'un nœud source. Le principe du parcours en largeur est le suivant : on explore en premier le nœud source, puis ses successeurs, puis les successeurs de ses successeurs, etc. Cet algorithme visite tous les nœuds du graphe atteignable à partir du nœud source. Il peut être utilisé afin de calculer les plus courts chemins entre le sommet source et tous les sommets du graphe. Sa complexité temporelle dans le pire cas est O(s+a) avec s le nombre de sommets et a le nombre d'arcs.

2) L'algorithme de parcours en profondeur: Le parcours en profondeur est un algorithme récursif de parcours de graphe ou d'arbre. A partir d'un sommet source, il explore un chemin dans le graphe jusqu'à arriver à une impasse ou à un nœud déjà visité. Il revient alors au dernier sommet pouvant mener à un autre chemin et explore celui-ci. L'algorithme s'arrête lorsque tous les sommets atteignables à partir du sommet source ont été visités. Il est utilisé afin de trouver l'ensemble des sommets accessibles depuis un sommet source. Sa complexité dans le pire des cas est O(s+a) avec s le nombre de sommets et a le nombre d'arcs.

3) L'algorithme de Dijkstra: Cet algorithme est un algorithme de recherche du plus court chemin dans un graphe pondéré positivement. Il est fondé sur un parcours en largeur. A partir d'un point de départ, il recherche le plus court chemin menant à la destination de façon itérative. L'algorithme se base sur le principe que tous les sous-chemins C-D du chemin le plus court entre les points A et B sont aussi le plus court chemin entre les points C et D. La complexité temporelle de l'algorithme de Dijkstra varie en fonction de la manière dont il est implémenté : avec un tas binaire, elle est de O((a+s) x log(s)) ; avec un tas de Fibonacci, elle est de O(a+s x log(s)).

4) L'algorithme A:* L'algorithme A* est un algorithme de recherche du plus court chemin d'un point à un autre dans un graphe. C'est est une version améliorée de l'algorithme de Dijkstra qui utilise des heuristiques pour orienter sa recherche du plus court chemin. Les heuristiques estiment la distance qui sépare le point considéré du point d'arrivée. Pour que l'algorithme soit admissible, c'est-à-dire pour qu'il garantisse de trouver le plus court chemin, l'heuristique choisie ne doit pas surestimer la distance au point d'arrivée. La complexité de l'algorithme dépend de la qualité de l'heuristique utilisée. L'un des avantages de A* est sa simplicité de calculs. Toutefois, celui-ci ne peut pas adapter le chemin trouvé si l'environnement (le graphe) change.

5) L'algorithme D ou l'algorithme A* dynamique:* L'algorithme D* ou l'algorithme A* dynamique. Cet algorithme se comporte comme A* avec la différence que l'algorithme permet une replanification rapide de la trajectoire si un obstacle inattendu est détecté sur la route. L'algorithme commence à étudier les nœuds à partir du nœud d'arrivée. Chaque nœud possède un "backpointer" qui représente le nœud étudié précédemment (donc le prochain nœud de la trajectoire finale), et le coût pour aller d'un nœud au nœud d'arrivée est connu. L'algorithme se termine lorsque le nœud de départ est rencontré. Le chemin à suivre est alors indiqué par les "backpointers".

6) LPA ou Lifelong Planning A*:* Cet algorithme est une version incrémental de A* qui réutilise des informations calculées lors de précédentes recherches. Sa première recherche est l'algorithme A*. L'algorithme trouve de manière répétée les chemins les plus courts d'un nœud de départ à un nœud d'arrivée dans un graphe, à mesure que des arcs ou des nœuds sont ajoutés ou supprimés ou que les coûts des arêtes sont modifiés. LPA* permet de réduire fortement la durée de recherche lorsque des changements mineurs

surviennent dans le graphe. En effet, l'algorithme ne réévalue que les noeuds affectés par le changement, contrairement à A*.

7) D Lite:* Cet algorithme est un algorithme de recherche heuristique incrémentale qui a le même comportement que l'algorithme D* mais qui se base sur le LPA*. Contrairement au LPA* qui commence sa recherche à partir du noeud de départ, D* Lite commence à chercher à partir du nœud d'arrivée. Cet algorithme, plus simple à implémenter que D* et a une performance égale voire supérieure à celui-ci.

B. L'algorithme choisi

Dans l'étude de notre robot, A* peut représenter un bon choix en raison de sa simplicité de calcul. Toutefois, l'algorithme présente un inconvénient : il est incable de s'ajuster lorsque des obstacles inattendus apparaissent sur le chemin. Il n'est donc pas adapté à notre situation. Parmi les algorithmes de recherche incrémentale, D*, LPA* et D* Lite, nous avons décidé d'utiliser le l'algorithme D* Lite, de par sa simplicité d'implémentation et son efficacité.

V. Suivi de trajectoire

A. Calcul de l'orientation

Afin de suivre la trajectoire du robot, il est essentiel de pouvoir calculer la rotation nécessaire pour qu'il se dirige vers ses objectifs. Pour ce faire, nous pouvons modéliser le problème de la manière suivante.

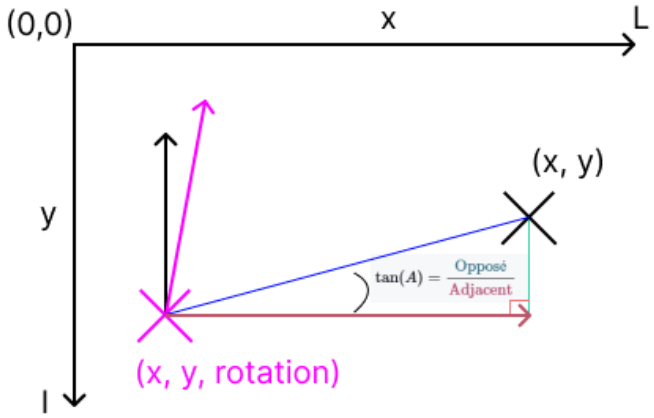

Fig. 7. Modélisation du problème du calcul de l'orientation

Il est nécessaire de prendre en compte les différents cas possibles lorsque le point (x,y) se situe en dessous du robot ou à sa gauche. En effet, cela permet de définir quatre quadrants distincts.

Le premier quadrant est déjà pris en compte dans la modélisation du problème, si l'on se place dans notre référentiel d'angles (voire figure 6) Nous pouvons récupérer la valeur correspondante de manière efficace en procédant comme suit:

(angle = valeur calculée par le premier quadrant)

1) Quadrant haut gauche:

$$res = (angle - 180) * -1$$

2) Quadrant bas gauche:

$$res = angle - 180$$

3) Quadrant bas droite:

$$res = angle * -1$$

VI. Conclusion

The conclusion goes here.

REMERCIEMENT

Un grand remerciement à M. Boumaza qui nous a encadrés, accueillis dans ses locaux et conseillés de manière pédagogique. Grâce à lui, cette expérience a été très enrichissante. Nous tenons également à remercier Télécom Nancy pour être à l'origine de cette collaboration, ainsi que les lecteurs de cet article pour leur attention et le temps accordé.

REFERENCES

[1] Mordechai Ben-Ari, Francesco Mondada, *Elements of Robotics* Livre sur SpringerLink, 2018.
[2] G.W. Lucas *trajectory Model for the Differential Steering System* Rossum project, 2000.
[3] Udek and Jenkin *Computational Principles of Mobile Robotics* Université de Colombia
[4] Tim Wescott *PID Without a PhD* Wescott Design Services, 2018
[5] Iconnue *Kinematics and Odometry for a Differential Drive Vehicle* California Institute of Technology
[6] Lulu *Modèle géométrique d'un robot mobile à roues différentielles* Le blog de Lulu, 2019