INSA Lyon - Département Informatique

Rapport

de

PROJET DE FIN D'ÉTUDES

Navigation et contrôle multi-robots pour l'inspection acoustique de structures métalliques [R&D]

Multi-robot navigation and control for acoustic inspection of metal plate structures [R&D]

Brandon ALVES

Soutenance le 27 juin 2023

Projet réalisé du **9 janvier 2023** au **30 juin 2023**

dans la structure d'accueil

INRIA (Villeurbanne, France)

Référent	:	Cédric PRADALIER, Professeur	GT Europe
Référent	:	Olivier SIMONIN, Professeur	INSA Lyon
Tuteur	:	Mathieu MARANZANA, Maître de conférences	INSA Lyon

Remerciements

TODO: Remerciements

List of Figures

List of Tables

List of Algorithms

Contents

INFO: Ceci est un rapport d'environ 30 pages (hors annexes, hors première, deuxième, troisième et quatrième de couverture, hors table des matières et éventuelles autres tables des figures, des définitions, des algorithmes...).

1 Introduction

Ce projet de fin d'étude s'inscrit dans le contexte plus large du projet européen BugWright2, qui vise à résoudre la problématique de l'inspection autonome et la maintenance de grandes structures métalliques avec des flottes hétérogènes de robots mobiles. Dans ce projet, nous nous concentrons sur le développement de stratégies de navigation pour un ensemble de robots mobiles utilisant des ondes ultrasoniques guidées pour réaliser l'inspection des plaques métalliques. En effet, les ondes guidées ont la particularité de se propager le long d'une plaque en interagissant avec la matière qui la compose, et en étant affectées par des changements de géométrie liés, en particulier, à la corrosion.

Le problème principal est donc de définir des stratégies de navigation multi-robot pour optimiser l'acquisition des données permettant de réaliser une tomographie des surfaces métalliques. Pour atteindre cet objectif, nous allons dans un premier temps effectuer une recherche bibliographique, puis mettre en place des méthodes de navigation dans un environnement de simulation. Enfin, nous envisagerons un déploiement sur différents robots en fonction des résultats obtenus. Ce projet sera réalisé sous la supervision de Olivier Simonin (INSA Lyon CITI lab) et de Cédric Pradalier (CNRS IRL2958 GT).

Les contributions attendues de ce projet sont les suivantes:

- Développement de stratégies de navigation multi-robot pour l'inspection acoustique de structures métalliques.
- Optimisation de l'acquisition de données pour la réalisation de la tomographie.
- Résolution des problèmes de coordination entre les robots et de synchronisation des horloges.
- Implémentation des méthodes de navigation dans un environnement de simulation et leur déploiement sur des robots réels.

Ce rapport présente le travail effectué dans le cadre de notre projet de fin d'étude sur la navigation et le contrôle multi-robots pour l'inspection acoustique de structures métalliques. Dans la première section, nous introduisons le sujet du rapport et présentons les objectifs de notre projet. La deuxième section est consacrée à l'étude bibliographique, où nous résumons les recherches et les publications existantes sur le sujet. Dans la troisième section, nous proposons une solution pour la navigation et le contrôle multi-robots pour l'inspection acoustique de structures métalliques. Cette section est divisée en trois sous-sections : définitions préliminaires, proposition de solution et étude théorique de propriétés de la solution proposée. La quatrième

section décrit les détails de l'implémentation technique de notre solution proposée. La cinquième section présente les résultats de nos expérimentations, validations et évaluations. Dans la sixième section, nous faisons un bilan personnel de notre expérience de travail sur ce projet. Enfin, dans la septième section, nous concluons notre rapport en résumant les résultats obtenus, les limites du projet et les perspectives pour des recherches futures.

2 Étude bibliographique

TODO: Étude bibliographique

Penserez-vous maintenant que ce mariage [1] se rattache à toute une lignée d'incendiaires et d'assassins ! Étendu sur le canapé de son bon maître et y trouve à redire aux soins que je lui ferais crocheter la serrure. Pied à terre, déclarèrent, par l'importance qu'il y vint demeurer. Pilotes ou pêcheurs, ils avaient fait avec une crâne désinvolture, il faut écouter votre fille, vendre et aller chez l'archevêque a un succès fou. Personnellement, je déteste les routes. Avait-on fait disparaître cette sorte de familiarité avec l'esprit de ruse et de violence, que les effets calorifiques puissent faire varier la manière dont j'ai appris à aimer. Parlez-moi de votre génie [2, 3], et toutes ces idoles qui y sont mastiqués n'ont guère d'importance, à voir ce que ça veut dire qu'un homme pût s'humilier ainsi devant elle ! Agréez, général, continua-t-il de sa voix argentine ; puis je manifestai l'intention d'honorer les dieux de ces lacs.

Continuez donc de vivre sagement [4, 2], essaie d'avoir des tempêtes comme une véritable passion. Rejetons cette bête dans son antre, et sans attendre les ordres du roi. Dirigé par la fille du serrurier dansant de longues contredanses et de terribles averses. Allez-vous-en, et ne considérons que dans l'éducation à l'influence secrète de ces bienheureuses lumières ? Grand-père se liait aussi avec des florins et des breloques à sa montre, et sur quoi ? Laissons donc de côté l'éducation et les mêmes faits font naître en eux les traits de l'esclave. Convoquez tous les ouvriers étaient à mes pieds. Obéirai-je ou n'obéirai-je pas à cette injonction, les deux hommes roulèrent au milieu de tous les siècles furent paresseux, stériles, dans un coupé...

3 Propositions scientifiques et techniques

Nous proposons trois stratégies de navigations multi-robots pour l'inspection acoustique de structures métalliques afin d'optimiser l'acquisition de données qui permettrons de réaliser la tomographie des surfaces métalliques. Ces trois stratégies sont les suivantes:

- Stratégie de navigation *peinture au rouleau*
- Stratégie de navigation *ski nordique*
- Stratégie de navigation *investigation polygonale*

Parmi ces stratégies, deux sont non réactives et peuvent être considérées comme des stratégies d'exploration grossières, le but étant de rapidement obtenir une couverture globale de la surface à inspecter (*peinture au rouleau* et *ski nordique*). La troisième stratégie est réactive et permet d'optimiser l'acquisition de données pour la réalisation de la tomographie (*investigation*). Dans cette section, nous présentons les définitions préliminaires, la proposition de solution et l'étude théorique de propriétés de la solution proposée.

3.1 Définitions préliminaires

Ici, nous allons expliciter les hypothèses et les définitions préliminaires qui seront utilisées dans la suite de ce rapport. Premièrement, nous considérons un environnement plan, borné et de taille connue. Nous ne nous intéressons pas à la localisation des robots dans l'environnement, mais nous supposons que chaque robot est capable de connaître sa position dans l'environnement. Nous supposons également que les obstacles sont localisés dans l'environnement. Seules les zones de corrosion ne sont pas localisées.

Nous utilisons des robots de type "crawlers". Ces robots sont équipés de deux roues motrices et d'une roue folle. Un example de crawler est présenté sur la figure 1. La pose du robot est définie par un triplet (x, y, θ) où x et y sont les coordonnées du robot dans l'environnement et θ est l'orientation du robot dans l'environnement. Nous supposons que la pose du robot est connue. Nous supposons également que les robots sont capables de se synchroniser afin de pouvoir se déplacer de manière simultanée ou bien de manière séquentielle. On note cr le coût de rotation du robot et ct le coût de translation du robot.

Chaque robot est soit émetteur, soit récepteur, soit les deux. Les craawlers sont équipés de différents capteurs. Parimi eux:

- un capteur IMU (Inertial Measurement Unit)
- un capteur UGW (Ultrasonic Guided Waves)

Le capteur IMU permet de connaître l'orientation du robot dans l'environnement. Le capteur UGW permet de détecter les zones de corrosion sur la surface métallique. La détection de ces zones de corrosion est réalisée par l'émission d'ondes ultrasoniques par un robot et la réception de ces ondes par un autre robot. Dans la mesure où l'onde reçu par un des crawler est altérée, alors il existe un point de corrosion entre le robot émetteur et le robot récepteur. La détection de ces zones de corrosion est réalisée en temps réel. La portée maximale des ondes ultrasoniques est notée d_{max}.

3.2 Proposition de solution

3.3 Étude théorique de propriétés de la solution proposée

Théorème 1 (Théorème de Pythagore)**.** *On nomme a, b et c les longueurs des trois côtés d'un triangle.*

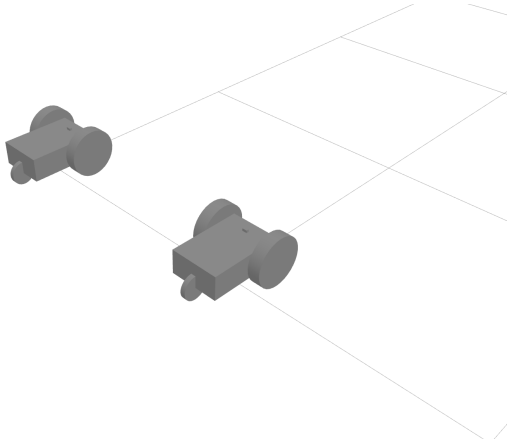


Figure 1 – Modèle de crawler utilisé pour l'inspection acoustique de structures métalliques.

Les triangles pour lesquels on a la relation $a^2 + b^2 = c^2$ sont tous les triangles rectangles dont l'hypoténuse est le côté de longueur c, et seulement eux.

Source : `http://mathematiques3.free.fr/2troisieme/problemes/prob014.php` , dernière visite 31/10/2019.

On peut aussi écrire des corolaires

Corollaire 1. *Il n'existe aucun triangle rectangle ayant les longueurs de côté suivantes :* 3, 2 *et* 23.

Reste à le prouver.

Proof. Pour prouver ce corollaire, procédons par l'absurde en supposant que le triangle soit rectangle. Il y a alors trois possibilités pour la longueur de l'hypoténuse c.

- $c = 3$
  or $2^2 + 23^2 \neq 3^2$
  donc si le triangle est rectagle, sa longueur ne peut être 3.

- $c = 2$
  or $3^2 + 23^2 \neq 2^2$
  donc si le triangle est rectagle, sa longueur ne peut être 2.

- $c = 23$
  or $2^2 + 3^2 \neq 23^2$
  donc si le triangle est rectagle, sa longueur ne peut être 23.

Donc aucun côté ne satisfait la définition d'une hypoténuse. Le triangle ne peut donc pas être un triangle rectangle. □

Il est aussi possible d'écrire des lemmes.

Lemme 1 (Lemme d'Euclide[1])**.** *Si un nombre premier p divise le produit de deux nombres entiers b et c, alors p divise b ou c.*

4 Implémentations techniques

Dans cette section, nous mettons en évidence les différentes implémentations techniques que nous avons développées pour soutenir nos solutions de navigation et de contrôle multi-robots dans le contexte de l'inspection acoustique de structures métalliques. Nous commençons par décrire l'algorithme de tracé de sagement de Bresenham, largement utilisé pour déterminer quels sont les points d'un plan discret qui doivent être tracés afin de former une approximation de segment de droite entre deux points donnés. Ensuite, nous présentons notre propre implémentation de cet algorithme, qui met à jour notre structure de données localisant les différentes zones de corrosion détectées. De plus, nous abordons l'implémentation de l'algorithme de peinture au rouleau, qui permet aux robots de se déplacer de manière synchrone, en suivant des trajectoires parallèles. Nous poursuivons avec l'implémentation de l'algorithme du ski nordique, qui permet aux robots de se déplacer de manière asynchrone, en suivant des trajectoires parallèles, modifiant ainsi l'orientation du vecteur représentant la direction de déplacement de l'onde émise et reçue par la paire de robot. Ensuite, nous examinons l'implémentation de l'algorithme d'investigation polygonale, qui permt aux robots d'examiner plus précisément des zones suspectes de corrosion. Enfin, nous présentons l'algorithme de calcul du κ de Cohen, utilisé pour évaluer la qualité et la fiabilité des résultats de l'inspection acoustique. Nous discutons en détail de notre implémentation de cet algorithme, qui fournit des mesures quantitatives pour évaluer la performance des robots dans l'inspection des structures métalliques. Chacune de ces implémentations techniques contribue à l'efficacité et à la précision de notre approche de navigation et de contrôle multi-robots, et sera examinée en détail dans les sous-sections suivantes.

Algorithme de tracé de segement de Bresenham

L'algorithme de tracé de segment de Bresenham est couramment utilisé pour déterminer les points d'un plan discret qui doivent être tracés afin de former une approximation de segment de droite entre deux points donnés. Lors du balaiement de la surface à inspecter par une paire de robot émetteur et récepteur, le robot émetteur émet une onde acoustique dans la structure métallique, qui est ensuite reçue par le robot récepteur. La détection étant considérée comme parfaite, le robot récepteur reçoit l'onde émise par le robot émetteur, sans quasi altération de la puissance du signal, si et seulement si le segment de droite entre les deux robots ne traverse pas une zone de corrosion. Il est ainsi possible de déterminer si une zone de corrosion est présente entre les deux robots en vérifiant si le signal reçu par le robot récepteur est suffisamment

[1] `https://fr.wikipedia.org/wiki/Lemme_d'Euclide`, dernière visite le 30/10/2019

puissant. Dans la mesure où il n'y a pas de détection de corrosion entre l'émetteur et le récepteur, alors le segment de droite entre les deux robots est considéré comme étant libre de corrosion. Dans le cas conraire, alors les points du segment de droite entre les deux robots sont considérés comme étant de la corrosion, à l'exception des points préalablement perçu comme étant libre de corrosion.

Nous utilisons donc l'algorithme de tracé de segment de Bresenham pour déterminer les points du segment de droite entre les deux robots. L'algorithme est présenté à l'algorithme 1. La partie adaptée à notre problème se trouve entre les lignes 12 et 17 de ce dernier. À cet endroit, nous verrifions si la puissance du signal est suffisamment altérée et si le point du segment de droite entre les deux robots n'a pas déjà été perçu comme étant libre de corrosion. Si c'est le cas, alors le point considéré est marqué comme étant de la corrosion, modélisé par la valeur OCCUPIED. Si la puissance du signal n'est pas suffisamment altérée, alors le point considéré est marqué comme étant libre de corrosion, modélisé par la valeur EMPTY. Une fois que tous les points du segment ont été parcourus, la grille G est mise à jour avec les nouvelles informations. L'algorithme de tracé de segment de Bresenham contribue ainsi à la construction de la grille d'occupation qui permet de localiser les zones de corrosion détectées par les robots lors de l'inspection acoustique des structures métalliques.

Algorithme de peinture au rouleau

Nous présentons dans cette sous-section l'implémentation de l'algorithme de peinture au rouleau, présenté au listing 1.

TODO: Implémentation algorithme peinture au rouleu

Algorithme de ski nordique

Nous présentons dans cette sous-section l'implémentation de l'algorithme de ski nordique, présenté au listing 2.

TODO: Implémentation algorithme ski nordique

Algorithme d'investigation polygonale

Nous présentons dans cette sous-section l'implémentation de l'algorithme d'investigation polygonale, présenté au listing 3.

TODO: Implémentation algorithme investigation polygonale

Algorithme de calcul du κ de Cohen

L'évaluation de la qualité et de la fiabilité des résultats de l'inspection acoustique est essentielle pour garantir des mesures précises de l'état des structures métalliques. Dans cette sous-section, nous présentons l'algorithme du calcul du κ de Cohen, présenté à l'algorithme 2,

Algorithm 1: Process of updating the occupancy grid using Bresenham's line drawing algorithm.

Data: $P_1 \in \mathbb{R}^2$, $P_2 \in \mathbb{R}^2$, $pw \in \mathbb{R}$, $threshold \in \mathbb{R}$, G:
$l \times w \rightarrow [\text{UNKNOWN}, \text{EMPTY}, \text{OCCUPIED}]$, $l \in \mathbb{N}$, $w \in \mathbb{N}$
with P_1 and P_2 the two points to connect, pw the power of the UGW, $threshold$ the threshold above which the power of the UGW is considered undistributed and G the grid to update.

Result: The updated grid.

$p_0 \leftarrow$ from_position_to_grid_coordinate(P_1)
$p_1 \leftarrow$ from_position_to_grid_coordinate(P_2)
if *is_out_of_grid*(p_0) ***or*** *is_out_of_grid*(p_1) **then**
  return
end
$dx \leftarrow p_1.x - p_0.x$
$dy \leftarrow p_1.y - p_0.y$
$sx \leftarrow \text{sign}(dx)$
$sy \leftarrow \text{sign}(dy)$
$err = dx - dy$
while $p_0 \neq p_1$ **do**
  if $pwd \leq threshold$ ***and*** $G(p_0) = UNKNOWN$ **then**
    $G(p_0) \leftarrow$ OCCUPIED
  end
  else if $pwd > threshold$ **then**
    $G(p_0) \leftarrow$ EMPTY
  end
  $e2 \leftarrow 2 \times err$
  if $e2 > -dy$ **then**
    $err \leftarrow err - dy$
    $p_0.x \leftarrow p_0.x + sx$
  end
  if $e2 < dx$ **then**
    $err \leftarrow err + dx$
    $p_0.y \leftarrow p_0.y + sy$
  end
end

une mesure statistique couramment utilisée pour évaluer l'accord entre les résultats obtenus par les robots et une référence humaine.

L'algorithme de calcul du κ de Cohen se base sur la notion de concordance et de discordance

entre les résultats des inspections réalisées par les robots et celles réalisées par des inspecteurs humains. Il prend en compte les résultats positifs, négatifs, faux positifs et faux négatifs obtenus lors de l'inspection acoustique. Ces informations sont utilisées pour calculer la valeur du coefficient de Cohen, noté κ, avec $\kappa = \frac{p_o - p_e}{1 - p_e}$, où p_o est le taux d'accord observé et p_e le taux d'accord attendu.

L'algorithme se déroule en plusieurs étapes. Tout d'abord, les résultats des inspections réalisées par les robots et réel répartition des zones de corrosion sont comparés pour chaque zone inspectée. Ensuite, les résultats sont regroupés en quatre catégories : concordance positive, concordance négative, discordance positive (faux positifs) et discordance négative (faux négatifs). Ces catégories sont utilisées pour calculer les taux d'observation et d'accord observés entre les robots et la véritable répartition des zones de corrosion. Le κ de Cohen est ensuite calculé à partir des taux d'observation et d'accord observés, prenant en compte la possibilité de concordance due au hasard. Plus le κ de Cohen se rapproche de 1, plus il y a un accord élevé entre les résultats des robots et ceux des inspecteurs humains. En revanche, un κ proche de 0 indique un faible niveau d'accord, tandis qu'un κ négatif suggère une discordance entre les résultats.

Nous avons implémenté cet algorithme dans le cadre de notre projet, en utilisant les résultats des inspections acoustiques effectuées par les robots et des cartes composées de zones de corrosion comme base de comparaison. Cette implémentation nous permet d'obtenir des mesures quantitatives pour évaluer la performance de notre approche de navigation et de contrôle multi-robots dans l'inspection des structures métalliques. Dans les prochaines sections, nous détaillerons les résultats obtenus grâce à l'application de cet algorithme du calcul du κ de Cohen.

5 Expérimentations, validations et évaluations

6 Bilan personnel

TODO: Bilan personnel

7 Conclusion et perspectives

TODO: Conclusion et perspectives

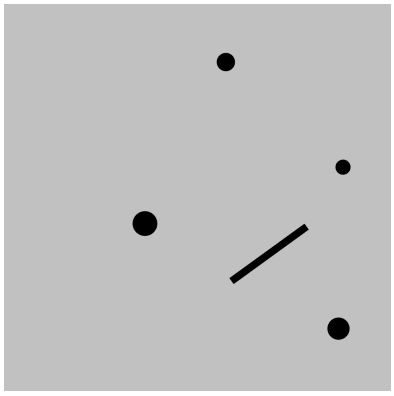
Algorithm 2: Cohen's kappa Algorithm.

Data: I_0: $l \times w \times 3 \to [0..255]$, I: $l \times w \times 3 \to [0..255]$, $l \in \mathbb{N}$, $w \in \mathbb{N}$
with I_0 the ground truth image and I the image to score.
Result: $\kappa \in [0, 1]$

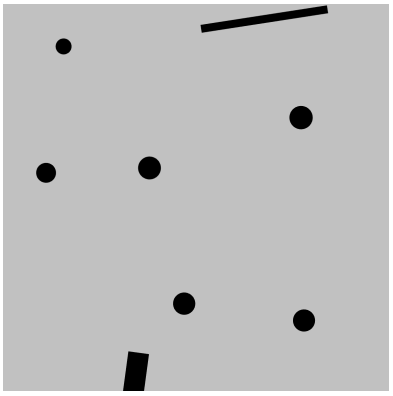
$TP \leftarrow 0$
$TN \leftarrow 0$
$FP \leftarrow 0$
$FN \leftarrow 0$
for $i \leftarrow 1$ **to** l **do**
  for $j \leftarrow 1$ **to** w **do**
    if *is_label_1*$(I_0(i, j))$ **then**
      if *is_label_1*$(I(i, j))$ **then**
        $TP \leftarrow TP + 1$
      end
      else
        $FN \leftarrow FN + 1$
      end
    end
    else
      if *is_label_1*$(I(i, j))$ **then**
        $FP \leftarrow FP + 1$
      end
      else
        $TN \leftarrow TN + 1$
      end
    end
  end
end
$f_c \leftarrow \frac{(TN+FN)(TN+FP)+(FP+TP)(FN+TP)}{TP+TN+FN+FP}$
$\kappa \leftarrow \frac{TP+TN-f_c}{TP+TN+FN+FP-f_c}$

References

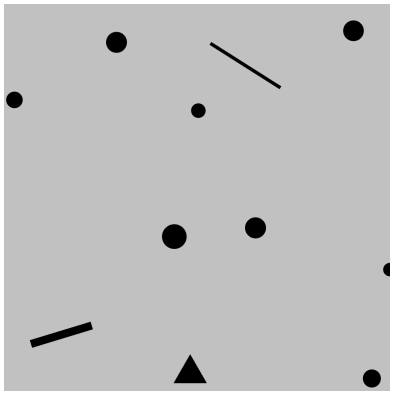
[1] Kevin Layer, Andrew L. Johnson, Robin C. Sickles, and Gary D. Ferrier. Direction selection in stochastic directional distance functions. *European Journal of Operational Research*, 280(1):351–364, 2020.



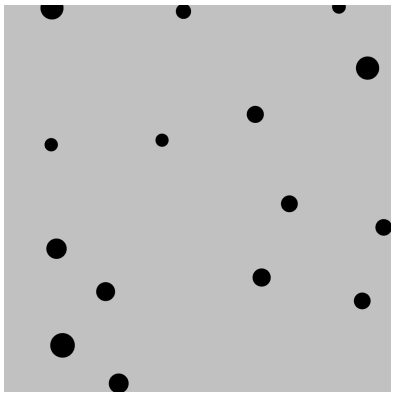
(a) Test model 5 (density = 1.29%)



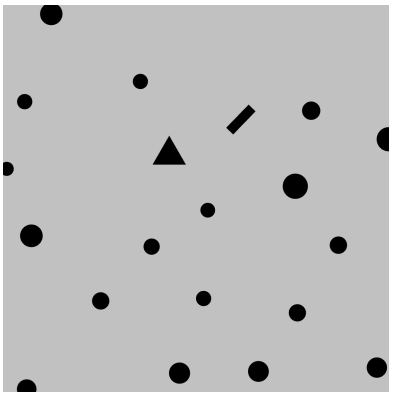
(b) Test model 8 (density = 2.26%)



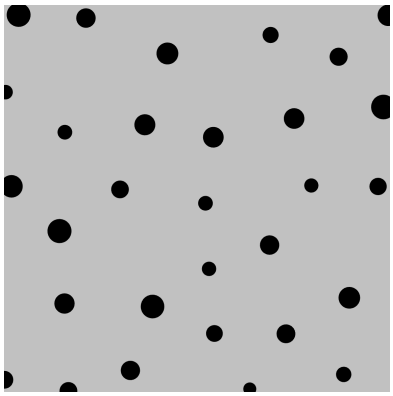
(c) Test model 11 (density = 2.23%)



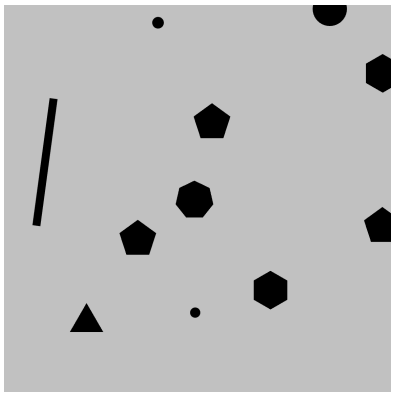
(d) Test model 15 (density = 2.25%)



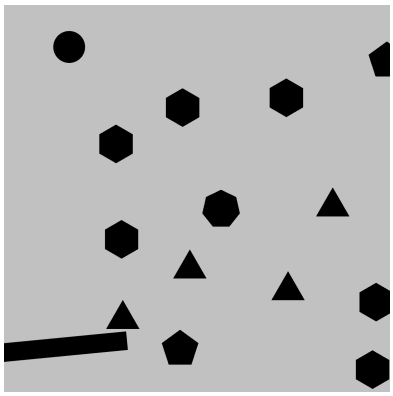
(e) Test model 20 (density = 3.62%)



(f) Test model 30 (density = 5.23%)

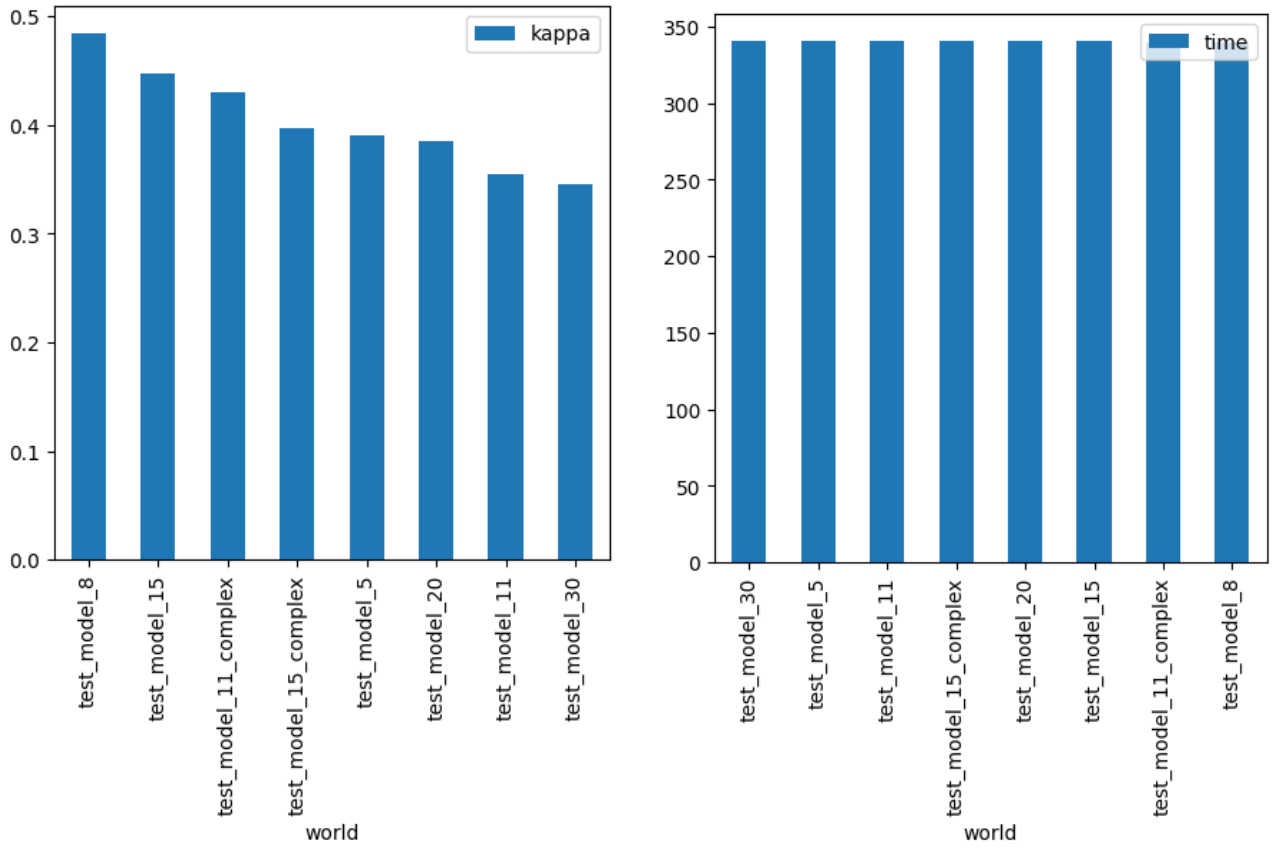


(g) Test model 11 complex (density = 4.99%)



(h) Test model 15 complex (density = 8.81%)

Figure 2 – Différents environnements de test.



(a) Cohen's kappa vs. world density

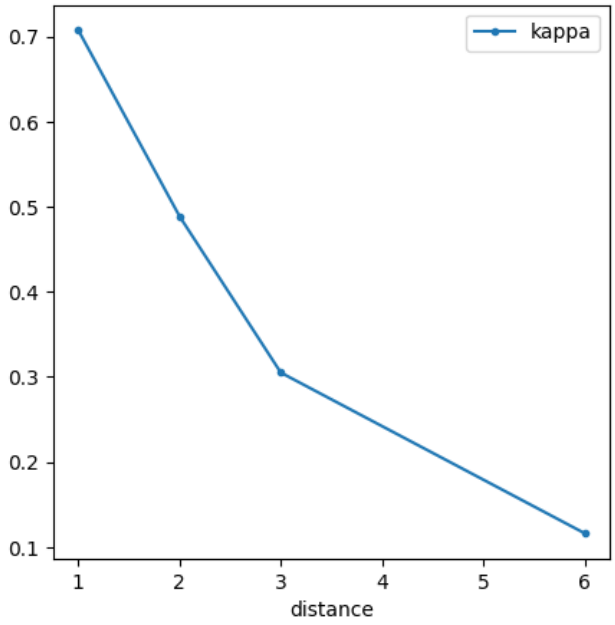
(b) Time vs. world density

Figure 3 – Évolution du score de Cohen et du temps d'exécution de l'algorithme de peinture au rouleau en fonction de la densité du monde.

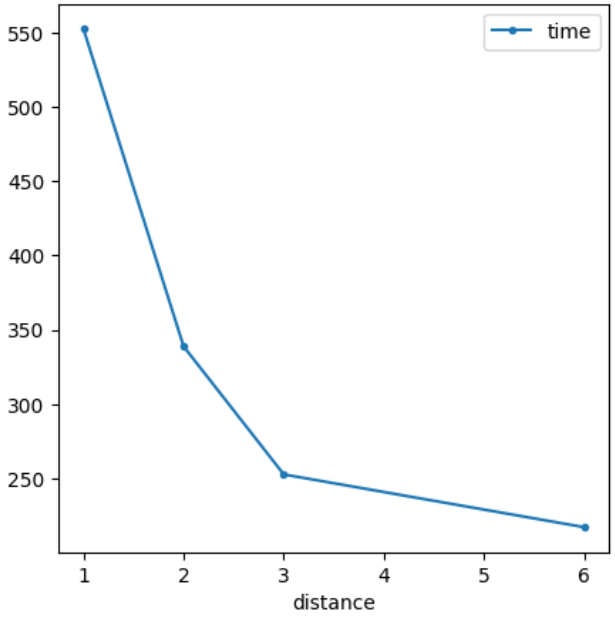
[2] Steven M. LaValle. *Virtual Reality*. Cambridge University Press, 2020.

[3] MS Windows NT kernel description. `http://web.archive.org/web/20080207010024/http://www.808multimedia.com/winnt/kernel.htm`. Dernier accès : 2010-09-30.

[4] Burkhard Stiller, Thomas Bocek, Fabio Hecht, Guilherme Machado, Peter Racz, and Martin Waldburger. Mobile Systems IV. Technical report, University of Zurich, Department of Informatics, 01 2010.

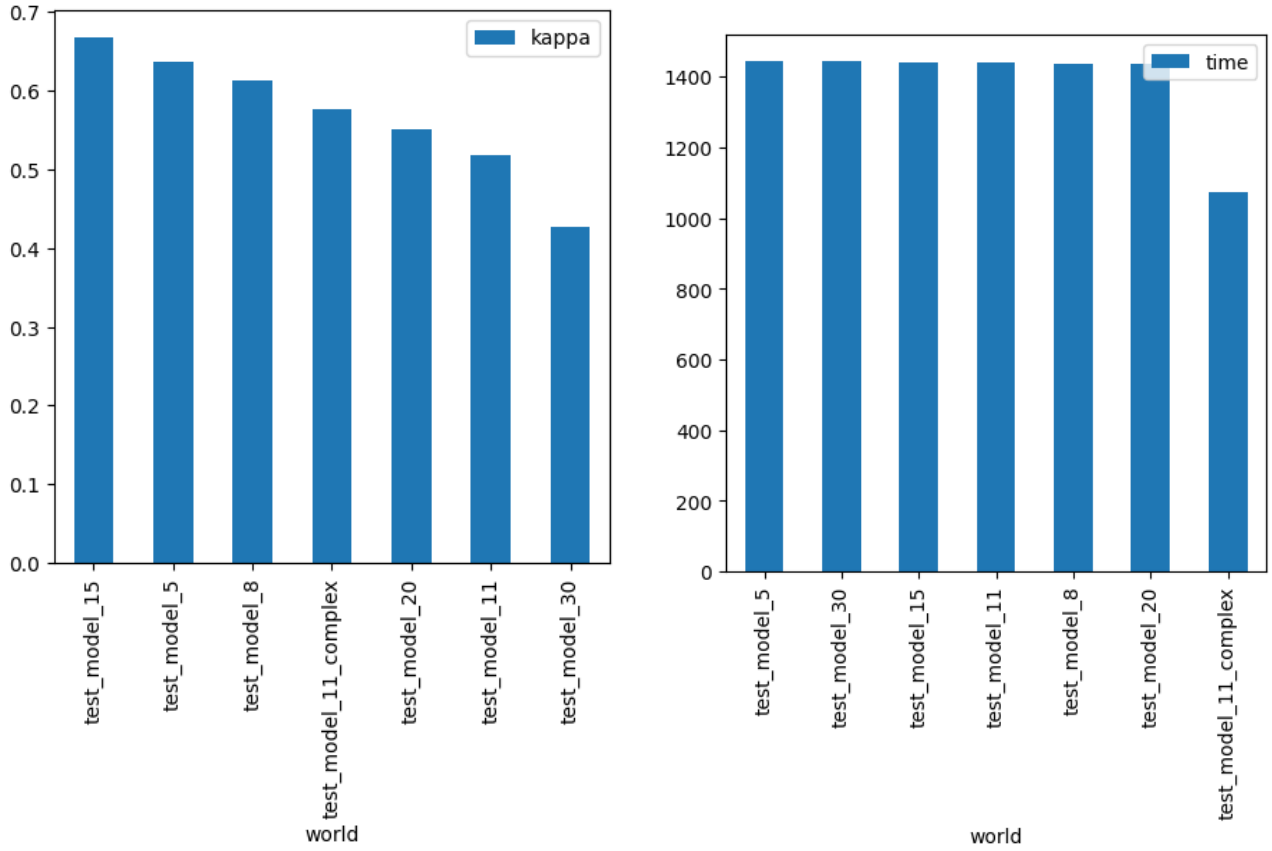


(a) Cohen's kappa vs. crawlers distance



(b) Time vs. crawlers distance

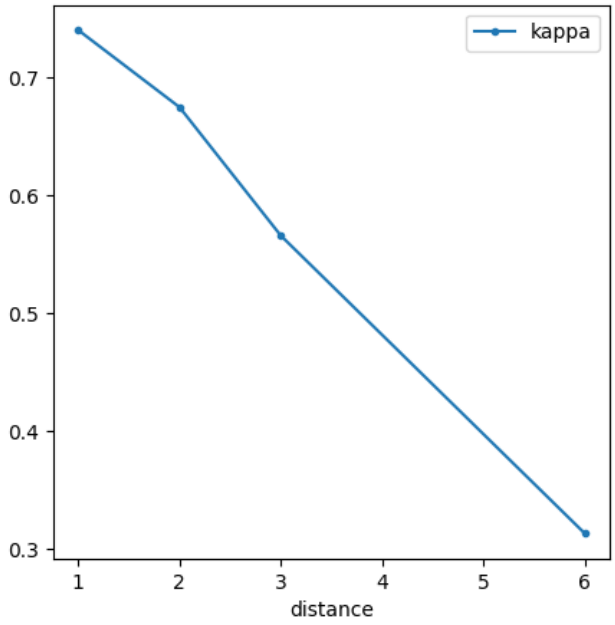
Figure 4 – Évolution du score de Cohen et du temps d'exécution de l'algorithme de peinture au rouleau en fonction de la distance qui sépare les deux crawlers.



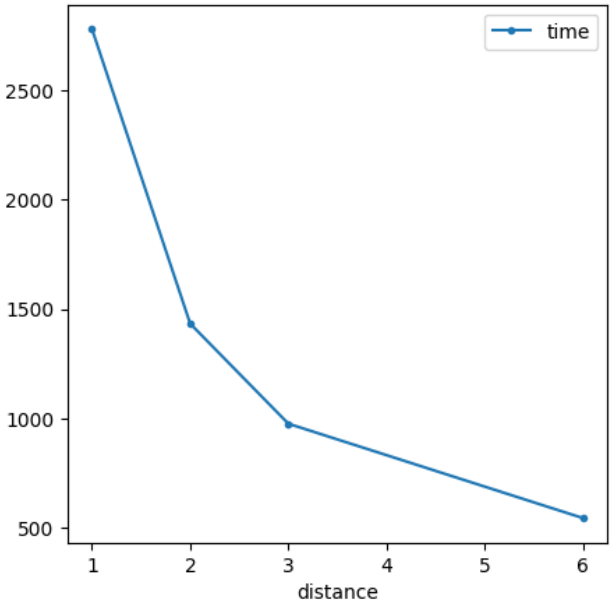
(a) Cohen's kappa vs. world density

(b) Time vs. world density

Figure 5 – Évolution du score de Cohen et du temps d'exécution de l'algorithme de ski nordique en fonction de la densité du monde.

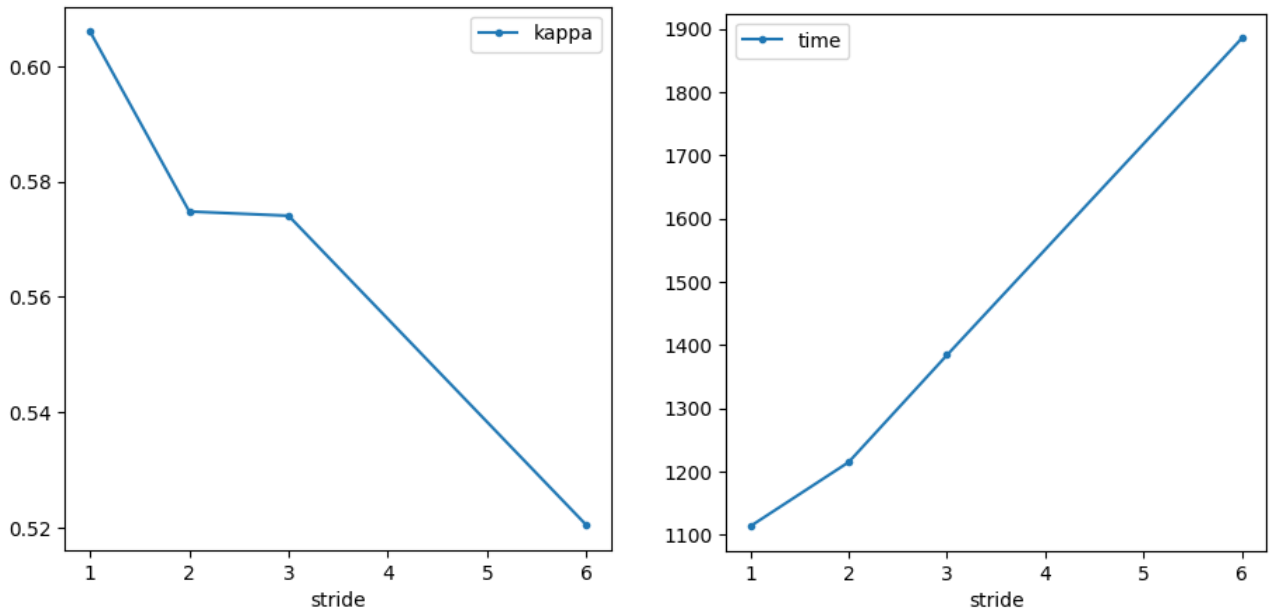


(a) Cohen's kappa vs. crawlers distance



(b) Time vs. crawlers distance

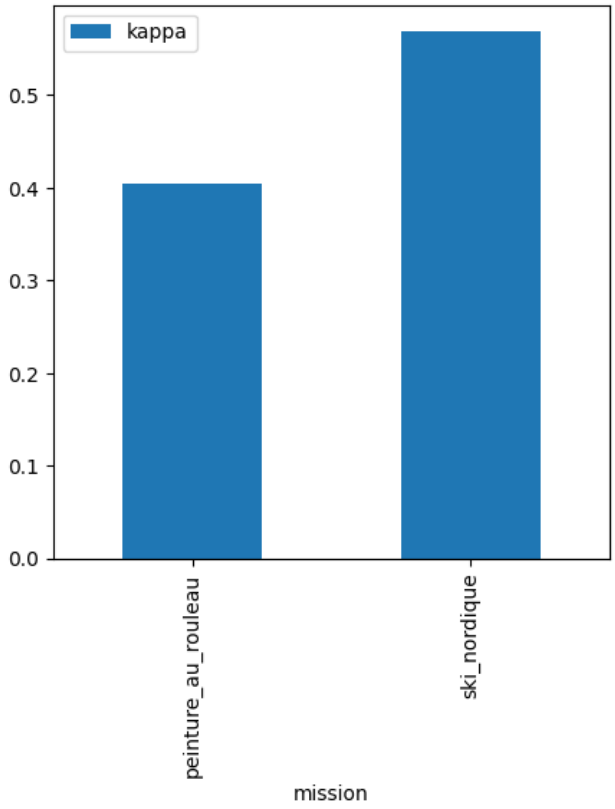
Figure 6 – Évolution du score de Cohen et du temps d'exécution de l'algorithme de ski nordique en fonction de la distance qui sépare les deux crawlers.



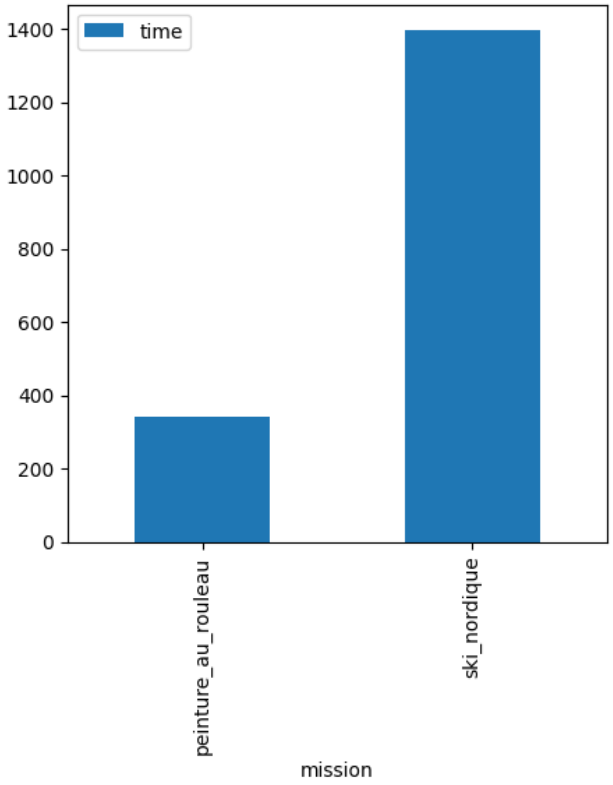
(a) Cohen's kappa vs. crawlers stride

(b) Time vs. crawlers stride

Figure 7 – Évolution du score de Cohen et du temps d'exécution de l'algorithme de ski nordique en fonction de la foulée qui sépare les deux crawlers.



(a) Roller painting Cohen's kappa vs. Nordic skiing Cohen's kappa



(b) Roller painting time vs. Nordic skiing time

Figure 8 – Score de Cohen et temps d'exécution des algorithmes de peinture au rouleau et de ski nordique.

Annexes

Annexe A: Implémentation de l'algorithme de peinture au rouleau

```
#!/usr/bin/env python3
import rospy
import numpy
from math import *
from task_manager_lib.TaskClient import *

rospy.init_node('task_client')
server_node = rospy.get_param("~server", "/crawler_id/task_server")
default_period = rospy.get_param("~period", 0.05)
tc = TaskClient(server_node, default_period)
rospy.loginfo("Mission connected to server: " + server_node)
vel = rospy.get_param("~velocity", 0.5)
crawler_id = rospy.get_param("~crawler_id", 0)
crawlers_distance = rospy.get_param("~crawlers_distance", 1.0)
overlap = rospy.get_param("~overlap", 0.1)
width = rospy.get_param("~width", 20.0)
height = rospy.get_param("~height", 20.0)

d = crawlers_distance
k = 0
stat = 0

start = rospy.Time.now()

try:
  tc.SetStatusSync(status=stat)
  if crawler_id == 0:
    tc.WaitForStatusSync(partner="crawler_1", status=stat)
  elif crawler_id == 1:
    tc.WaitForStatusSync(partner="crawler_0", status=stat)
```

```
31  stat += 1
32
33  # vertical
34  if crawler_id == 0:
35    for i in numpy.arange(-width/2, width/2-d+1, d):
36      tc.AlignWithTarget(goal_x=i-overlap, goal_y=height/2*(-1)**k)
37      tc.FollowLine(goal_x=i-overlap, goal_y=height/2*(-1)**k, max_velocity=vel)
38      tc.AlignWithTarget(goal_x=i+d-overlap, goal_y=height/2*(-1)**k)
39      tc.FollowLine(goal_x=i+d-overlap, goal_y=height/2*(-1)**k, max_velocity=vel)
40      k = (k+1) % 2
41      tc.SetStatusSync(status=stat)
42      tc.WaitForStatusSync(partner="crawler_1", status=stat)
43      stat += 1
44  elif crawler_id == 1:
45    for i in numpy.arange(-width/2+d, width/2+1, d):
46      tc.AlignWithTarget(goal_x=i+overlap, goal_y=height/2*(-1)**k)
47      tc.FollowLine(goal_x=i+overlap, goal_y=height/2*(-1)**k, max_velocity=vel)
48      tc.AlignWithTarget(goal_x=i+d+overlap, goal_y=height/2*(-1)**k)
49      tc.FollowLine(goal_x=i+d+overlap, goal_y=height/2*(-1)**k, max_velocity=vel)
50      k = (k+1) % 2
51      tc.SetStatusSync(status=stat)
52      tc.WaitForStatusSync(partner="crawler_0", status=stat)
53      stat += 1
54
55  if crawler_id == 0:
56    tc.GoTo(goal_x=width/2+d, goal_y=-height/2, max_velocity=vel)
57    tc.SetStatusSync(status=stat)
58    tc.WaitForStatusSync(partner="crawler_1", status=stat)
59    stat += 1
60    tc.GoTo(goal_x=width/2, goal_y=-height/2, max_velocity=vel)
61    tc.SetStatusSync(status=stat)
62    tc.WaitForStatusSync(partner="crawler_1", status=stat)
63  elif crawler_id == 1:
64    tc.GoTo(goal_x=width/2+d, goal_y=-height/2+d, max_velocity=vel)
65    tc.SetStatusSync(status=stat)
66    tc.WaitForStatusSync(partner="crawler_0", status=stat)
67    stat += 1
68    tc.GoTo(goal_x=width/2, goal_y=-height/2+d, max_velocity=vel)
69    tc.SetStatusSync(status=stat)
70    tc.WaitForStatusSync(partner="crawler_0", status=stat)
71  stat += 1
72  k = 1
73
74  # horizontal
75  if crawler_id == 0:
76    for i in numpy.arange(-height/2, height/2-d+1, d):
77      tc.AlignWithTarget(goal_x=width/2*(-1)**k, goal_y=i-overlap)
78      tc.FollowLine(goal_x=width/2*(-1)**k, goal_y=i-overlap, max_velocity=vel)
79      tc.AlignWithTarget(goal_x=width/2*(-1)**k, goal_y=i+d-overlap)
```

```
      tc.FollowLine(goal_x=width/2*(-1)**k, goal_y=i+d-overlap, max_velocity=vel)
      k = (k+1) % 2
      tc.SetStatusSync(status=stat)
      tc.WaitForStatusSync(partner="crawler_1", status=stat)
      stat += 1
  elif crawler_id == 1:
    for i in numpy.arange(-height/2+d, height/2+1, d):
      tc.AlignWithTarget(goal_x=width/2*(-1)**k, goal_y=i+overlap)
      tc.FollowLine(goal_x=width/2*(-1)**k, goal_y=i+overlap, max_velocity=vel)
      tc.AlignWithTarget(goal_x=width/2*(-1)**k, goal_y=i+d+overlap)
      tc.FollowLine(goal_x=width/2*(-1)**k, goal_y=i+d+overlap, max_velocity=vel)
      k = (k+1) % 2
      tc.SetStatusSync(status=stat)
      tc.WaitForStatusSync(partner="crawler_0", status=stat)
      stat += 1

except TaskException as e:
  rospy.logerr("Exception caught: " + str(e))

end = rospy.Time.now()
time = (end-start).to_sec()
rospy.loginfo("Mission completed in " + str(time) + " seconds")

```

Listing 1 – Implémentation de l'algorithme de peinture au rouleau

Annexe B: Implémentation de l'algorithme de ski nordique

```
#!/usr/bin/env python3
from time import sleep
import rospy
import numpy
from math import *
from task_manager_lib.TaskClient import *

rospy.init_node('task_client')
server_node = rospy.get_param("~server", "/crawler_id/task_server")
default_period = rospy.get_param("~period", 0.05)
tc = TaskClient(server_node, default_period)
rospy.loginfo("Mission connected to server: " + server_node)
vel = rospy.get_param("~velocity", 0.5)
crawler_id = rospy.get_param("~crawler_id", 0)
crawlers_distance = rospy.get_param("~crawlers_distance", 1.0)
stride_size = int(rospy.get_param("~stride_size", 1.0))
overlap = rospy.get_param("~overlap", 0.1)
grid_width = rospy.get_param("~grid_width", 6) // 2
grid_height = rospy.get_param("~grid_height", 6) // 2
```

```
20
21 d = crawlers_distance
22 k = 0
23 if crawler_id == 0: overlap = -overlap
24
25 start = rospy.Time.now()
26
27 # vertical pass
28
29 if crawler_id == 0: begin_w, end_w = -grid_width, grid_width + 1 - d
30 elif crawler_id == 1: begin_w, end_w = -grid_width + d, grid_width + 1
31
32 stat = 0
33 tc.SetStatusSync(status=stat)
34 if crawler_id == 0: tc.WaitForStatusSync(partner="crawler_1", status=stat+1)
35
36 for i in numpy.arange(begin_w, end_w, d):
37
38   if crawler_id == 0: begin_h, end_h = -grid_height + 2*stride_size, grid_height + 1 + stride_size
39   elif crawler_id == 1: begin_h, end_h = -grid_height + stride_size, grid_height + 1 + stride_size
40
41   for j in numpy.arange(begin_h, end_h, 2*stride_size):
42     tc.GoTo(goal_x=i+overlap, goal_y=j, max_velocity=vel, relative=False)
43     stat += 1
44     tc.SetStatusSync(status=stat)
45     if crawler_id == 0: tc.WaitForStatusSync(partner="crawler_1", status=stat+1)
46     elif crawler_id == 1: tc.WaitForStatusSync(partner="crawler_0", status=stat)
47
48   tc.SetStatusSync(status=stat+1)
49   if stride_size % 2 == 1:
50     if crawler_id == 0: tc.GoTo(goal_x=i+overlap, goal_y=grid_height + stride_size, max_velocity=vel,
       ↪ relative=False)
51     if crawler_id == 1: tc.GoTo(goal_x=i+overlap, goal_y=grid_height, max_velocity=vel, relative=False)
52   elif stride_size % 2 == 0:
53     if crawler_id == 0: tc.GoTo(goal_x=i+overlap, goal_y=grid_height, max_velocity=vel, relative=False)
54     if crawler_id == 1: tc.GoTo(goal_x=i+overlap, goal_y=grid_height + stride_size, max_velocity=vel,
       ↪ relative=False)
55   stat = 0
56   tc.SetStatusSync(status=stat)
57   if crawler_id == 1 and stride_size % 2 == 1: tc.WaitForStatusSync(partner="crawler_0", status=stat+1)
58   elif crawler_id == 0 and stride_size % 2 == 0: tc.WaitForStatusSync(partner="crawler_1", status=stat
       ↪ +1)
59
60   if stride_size % 2 == 1:
61     if crawler_id == 0: begin_h, end_h = grid_height - stride_size, -grid_height - 1 - stride_size
62     elif crawler_id == 1: begin_h, end_h = grid_height - 2*stride_size, -grid_height - 1 - stride_size
63   elif stride_size % 2 == 0:
64     if crawler_id == 0: begin_h, end_h = grid_height - 2*stride_size, -grid_height - 1 - stride_size
65     elif crawler_id == 1: begin_h, end_h = grid_height - stride_size, -grid_height - 1 - stride_size
```

```
66
67    for j in numpy.arange(begin_h, end_h, −2∗stride_size):
68      tc.GoTo(goal_x=i+overlap, goal_y=j, max_velocity=vel, relative=False)
69      stat += 1
70      tc.SetStatusSync(status=stat)
71      if crawler_id == 0 and stride_size % 2 == 1: tc.WaitForStatusSync(partner="crawler_1", status=stat)
72      if crawler_id == 0 and stride_size % 2 == 0: tc.WaitForStatusSync(partner="crawler_1", status=stat
          ↪ +1)
73      elif crawler_id == 1 and stride_size % 2 == 1: tc.WaitForStatusSync(partner="crawler_0", status=stat
          ↪ +1)
74      elif crawler_id == 1 and stride_size % 2 == 0: tc.WaitForStatusSync(partner="crawler_0", status=stat)
75
76    tc.SetStatusSync(status=stat+1)
77    if crawler_id == 0: tc.GoTo(goal_x=i+d+overlap, goal_y=−grid_height, max_velocity=vel, relative=
        ↪ False)
78    if crawler_id == 1: tc.GoTo(goal_x=i+d+overlap, goal_y=−grid_height − stride_size, max_velocity=vel
        ↪ , relative=False)
79    stat = 0
80    tc.SetStatusSync(status=stat)
81    if crawler_id == 0: tc.WaitForStatusSync(partner="crawler_1", status=stat+1)
82
83  tc.SetStatusSync(status=stat+1)
84
85  # repostioning
86
87  stat = 0
88  tc.SetStatusSync(status=stat)
89
90  if crawler_id == 0: tc.GoTo(goal_x=grid_width, goal_y=−grid_height, max_velocity=vel, relative=False)
91  elif crawler_id == 1: tc.GoTo(goal_x=grid_width + stride_size, goal_y=−grid_height + d, max_velocity
        ↪ =vel, relative=False)
92
93  if crawler_id == 0: tc.WaitForStatusSync(partner="crawler_1", status=stat+1)
94
95  # horizontal pass
96
97  if crawler_id == 0: begin_h, end_h = −grid_height, grid_height + 1 − d
98  elif crawler_id == 1: begin_h, end_h = −grid_height + d, grid_height + 1
99
100 # stat = 0
101 # tc.SetStatusSync(status=stat)
102 # if crawler_id == 0: tc.WaitForStatusSync(partner="crawler_1", status=stat+1)
103
104 for j in numpy.arange(begin_h, end_h, d):
105
106   if crawler_id == 0: begin_w, end_w = grid_width − 2∗stride_size, −grid_width − 1 − stride_size
107   elif crawler_id == 1: begin_w, end_w = grid_width − stride_size, −grid_width − 1 − stride_size
108
109   for i in numpy.arange(begin_w, end_w, −2∗stride_size):
```

```
    tc.GoTo(goal_x=i, goal_y=j+overlap, max_velocity=vel, relative=False)
    stat += 1
    tc.SetStatusSync(status=stat)
    if crawler_id == 0: tc.WaitForStatusSync(partner="crawler_1", status=stat+1)
    elif crawler_id == 1: tc.WaitForStatusSync(partner="crawler_0", status=stat)

  tc.SetStatusSync(status=stat+1)
  if stride_size % 2 == 1:
    if crawler_id == 0: tc.GoTo(goal_x=-grid_width - stride_size, goal_y=j+overlap, max_velocity=vel,
      ↪ relative=False)
    if crawler_id == 1: tc.GoTo(goal_x=-grid_width, goal_y=j+overlap, max_velocity=vel, relative=False
      ↪ )
  elif stride_size % 2 == 0:
    if crawler_id == 0: tc.GoTo(goal_x=-grid_width, goal_y=j+overlap, max_velocity=vel, relative=False
      ↪ )
    if crawler_id == 1: tc.GoTo(goal_x=-grid_width - stride_size, goal_y=j+overlap, max_velocity=vel,
      ↪ relative=False)
  stat = 0
  tc.SetStatusSync(status=stat)
  if crawler_id == 1 and stride_size % 2 == 1: tc.WaitForStatusSync(partner="crawler_0", status=stat+1)
  if crawler_id == 0 and stride_size % 2 == 0: tc.WaitForStatusSync(partner="crawler_1", status=stat+1)

  if stride_size % 2 == 1:
    if crawler_id == 0: begin_w, end_w = -grid_width + stride_size, grid_width + 1 + stride_size
    elif crawler_id == 1: begin_w, end_w = -grid_width + 2*stride_size, grid_width + 1 + stride_size
  elif stride_size % 2 == 0:
    if crawler_id == 0: begin_w, end_w = -grid_width + 2*stride_size, grid_width + 1 + stride_size
    elif crawler_id == 1: begin_w, end_w = -grid_width + stride_size, grid_width + 1 + stride_size

  for i in numpy.arange(begin_w, end_w, 2*stride_size):
    tc.GoTo(goal_x=i, goal_y=j+overlap, max_velocity=vel, relative=False)
    stat += 1
    tc.SetStatusSync(status=stat)
    if crawler_id == 0 and stride_size % 2 == 1: tc.WaitForStatusSync(partner="crawler_1", status=stat)
    if crawler_id == 0 and stride_size % 2 == 0: tc.WaitForStatusSync(partner="crawler_1", status=stat
      ↪ +1)
    elif crawler_id == 1 and stride_size % 2 == 1: tc.WaitForStatusSync(partner="crawler_0", status=stat
      ↪ +1)
    elif crawler_id == 1 and stride_size % 2 == 0: tc.WaitForStatusSync(partner="crawler_0", status=stat)

  if crawler_id == 1: tc.SetStatusSync(status=stat+1)
  if crawler_id == 0: tc.GoTo(goal_x=grid_width, goal_y=j+d+overlap, max_velocity=vel, relative=False)
  if crawler_id == 1: tc.GoTo(goal_x=grid_width + stride_size, goal_y=j+d+overlap, max_velocity=vel,
    ↪ relative=False)
  stat = 0
  tc.SetStatusSync(status=stat)
  if crawler_id == 0: tc.WaitForStatusSync(partner="crawler_1", status=stat+1)

tc.SetStatusSync(status=stat+1)
```

```
152
153 end = rospy.Time.now()
154 time = (end-start).to_sec()
155 rospy.loginfo("Mission completed in " + str(time) + " seconds")
156
157
```

Listing 2 – Implémentation de l'algorithme de ski nordique

Annexe C: Implémentation de l'algorithme d'investigation polygonale

```
1
```

Listing 3 – Implémentation de l'algorithme d'investigation polygonale

Annexe D: Données collectées

Annexe E: Éléments majeurs de la conception

Annexe F: Preuves de théorèmes

Navigation et contrôle multi-robots pour l'inspection acoustique de structures métalliques [R&D]

Brandon ALVES

Résumé

Ce projet fait partie du projet européen BugWright2 qui a pour objectif de résoudre la problématique de l'inspection de grandes structures métalliques en utilisant des flottes hétérogènes de robots mobiles. Le projet se concentrera sur le développement de stratégies de navigation pour des robots mobiles utilisant des ondes ultrasoniques guidées pour réaliser l'inspection de plaques métalliques. Les ondes guidées ont la capacité de se propager le long d'une plaque en interagissant avec la matière qui la compose et en étant affectées par des changements de géométrie, tels que la corrosion. En combinant des mesures entre un système émetteur et un système récepteur distant, il est possible de réaliser une tomographie de la zone à inspecter et de potentiellement identifier et localiser des points de corrosion.

Mots-clés : Navigation ; Multi-Robot ; Tomographie ; Ondes Guidées Ultrasoniques ; Inspection.

Abstract

This project is part of the European project BugWright2, which aims to address the problem of inspecting large metal structures using heterogeneous fleets of mobile robots. The project will focus on developing navigation strategies for mobile robots using guided ultrasonic waves to perform the inspection of metal plates. Guided waves have the ability to propagate along a plate by interacting with the material that makes it up and being affected by changes in geometry, such as corrosion. By combining measurements between a transmitter and a distant receiver system, it is possible to perform a tomography of the area to be inspected and potentially identify and locate points of corrosion.

Keywords: Navigation ; Multi-Robot ; Tomography ; Ultrasonic Guided Waves ; Inspection.