



*TD 4 - Architecture Comportementale (AOC)*

*DI MARTINO Jonathan*

*EL ANSARI Fayssal*

*12-12-2023*

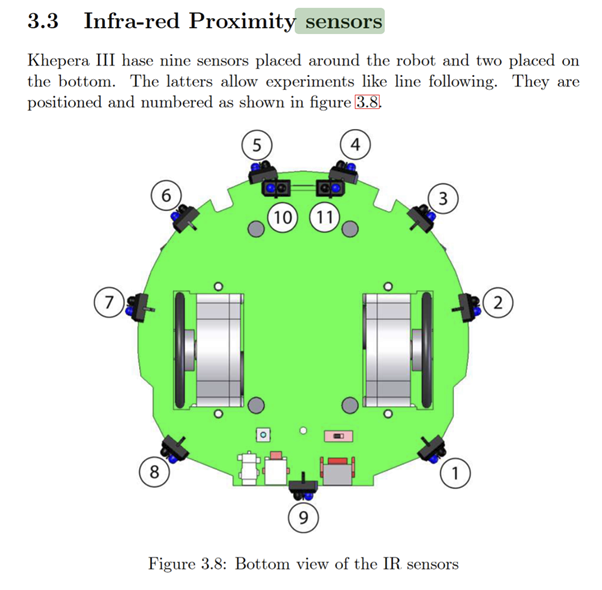
***Question 1 :*** *Une fois le simulateur et son IDE installés, vous pouvez parcourir l’exemple de la mise en œuvre du robot Khepera III pour vous assurer que tout est opérationnel.*

*Simulateur et IDE installés : ✅*

*Mise en oeuvre du robot Khepera III : ✅*

Tout est opérationnel.

***Question 2*** *: Reprenez votre TD précédent et implémentez le déplacement de votre robot en ligne droite et son arrêt devant le premier obstacle rencontré.*



D'après l'image, il n'est pas nécessaire d'utiliser tous les capteurs pour la détection, seulement ceux situés à l'avant, soit les capteurs de 2 à 7. Cependant, le simulateur Khepera utilisé dispose de 9 capteurs au total. Nous devons définir un seuil de valeur et le comparer aux valeurs rapportées par les capteurs. Nous commençons par ordonner au robot d'avancer à vitesse maximale, sachant que la valeur des capteurs est proportionnelle à la distance entre le robot et l'obstacle.

Pour identifier quels capteurs sont à l'avant, nous laissons le robot avancer jusqu'à ce qu'il soit bloqué, tout en imprimant les valeurs de tous les capteurs. À partir de ces données, nous pouvons déterminer quels sont les capteurs avant. Par exemple, considérons deux séquences consécutives de valeurs imprimées :

* Sensor 0 value: 10.300212
* Sensor 1 value: 12.259441
* Sensor 2 value: 528.153444
* Sensor 3 value: 3976.745031
* Sensor 4 value: 3962.206367
* Sensor 5 value: 1085.342326
* Sensor 6 value: 10.617176
* Sensor 7 value: 4.233496
* Sensor 8 value: 17.439022
* Sensor 0 value: 8.632479
* Sensor 1 value: 9.438946
* Sensor 2 value: 972.758053
* Sensor 3 value: 3971.042549
* Sensor 4 value: 3975.740341
* Sensor 5 value: 874.837765
* Sensor 6 value: 4.087047
* Sensor 7 value: 20.617325
* Sensor 8 value: 12.533534

Nous observons que les capteurs avec les valeurs les plus élevées sont les capteurs 2, 3, 4 et 5, avec des valeurs dépassant 100. Nous choisissons 20 comme seuil. Cela signifie que nous pouvons ignorer les autres capteurs. Il est important de noter que plus la valeur est élevée, plus l'obstacle est proche. Ainsi, si la valeur de l'un des capteurs est PLUS GRANDE que le seuil, le robot doit s'arrêter.

Nous avons remarqué que le robot ne s'arrête pas complètement et continue de faire de petits mouvements. Cela est dû au fait que les valeurs des capteurs ne sont pas exactes : il se peut qu'un capteur donne une valeur supérieure au seuil, mais lors de la prochaine itération de la boucle, la même valeur peut être inférieure au seuil. Pour corriger ce comportement, nous pourrions modifier l'algorithme pour sortir complètement de la boucle au lieu de continuer à vérifier si les conditions de distance sont respectées. Cependant, si l'environnement change et que l'obstacle n'est plus devant le robot, celui-ci ne pourra pas poursuivre sa route.

***Question 3*** *: Avec le simulateur Webots, en utilisant le robot khepera III, implémentez cet algorithme en utilisant tous les proximétres du robot. Choisissez les bons poids pour que le robot est un comportement satisfaisant.*

Pour implémenter l'algorithme d'évitement d'obstacle de Braitenberg dans Webots avec le robot Khepera III :

1. **Utilisation des Proximètres :** Le robot Khepera III est équipé de capteurs de proximité, ou proximètres, répartis autour de sa structure. Ces capteurs mesurent la distance entre le robot et les obstacles environnants. Leur rôle est de fournir des informations en temps réel sur l'environnement immédiat du robot.
2. **Principe de Braitenberg :** L'algorithme de Braitenberg est basé sur une idée simple mais puissante : les capteurs influencent directement les moteurs. Ainsi, la proximité d'un obstacle détectée par un capteur va affecter la vitesse du moteur correspondant.
3. **Calcul de Vl et Vr :** Pour le robot Khepera III, les vitesses des moteurs gauche (**Vl**) et droit (**Vr**) sont déterminées en fonction des lectures des capteurs. Plus précisément, **Vl** et **Vr** sont calculées en sommant les valeurs de tous les capteurs de chaque côté, chacune pondérée par un poids spécifique **Wi**. Ces poids déterminent l'influence relative de chaque capteur sur les moteurs.
4. **Ajustement des Poids (Wi) :** Les poids **Wi** sont des paramètres clés qui contrôlent la réactivité du robot aux obstacles. Un poids élevé signifie qu'un capteur a une grande influence sur le mouvement du robot. Ces poids peuvent être ajustés manuellement pour affiner le comportement du robot ou déterminés automatiquement par des méthodes d'apprentissage, comme des algorithmes génétiques.
5. **Implémentation dans la Boucle Principale :** Dans notre programme, nous lisons continuellement les valeurs des capteurs. Nous utilisons ensuite ces valeurs pour calculer **Vl** et **Vr** en appliquant la formule de Braitenberg. En fonction des détections des capteurs, le robot ajustera sa trajectoire, accélérant un moteur et ralentissant l'autre pour naviguer autour des obstacles.

En résumé, l'approche de Braitenberg, en intégrant l'interaction directe entre capteurs et moteurs, permet au robot Khepera III de répondre de manière dynamique à son environnement, démontrant ainsi l'efficacité et la polyvalence de principes simples dans la robotique avancée.

***Question 4*** *: Comment implémenteriez-vous un réseau de neurones pour l’évitement d’obstacles selon braintenberg dans le cas de l’alphabot2 ?*

Pour implémenter un réseau de neurones pour l'évitement d'obstacles selon le principe de Braitenberg sur l'Alphabot2, qui possède des détecteurs d'obstacles à sortie booléenne, nous devons adapter notre approche. Voici comment nous pouvons procéder :

1. **Structure du Réseau de Neurones :** Nous créerons un réseau de neurones simple sans couches cachées. Les entrées de ce réseau seront les signaux des détecteurs d'obstacle, et les sorties seront les vitesses des moteurs gauche (**Vl**) et droit (**Vr**).
2. **Traitement des Signaux Booléens :** Les détecteurs de l'Alphabot2 fournissent des signaux booléens, que nous traiterons comme des entrées binaires. Une valeur de **1** indiquerait la présence d'un obstacle, tandis qu'une valeur de **0** signifierait l'absence d'obstacle.
3. **Connexions et Poids :** Dans notre réseau, chaque détecteur sera connecté aux neurones de sortie. Les poids de ces connexions influenceront la vitesse des moteurs. Nous pouvons commencer avec des poids prédéfinis ou les ajuster à l'aide d'un processus d'apprentissage.
4. **Logique d'Évitement d'Obstacles :** Nous établirons une logique simple pour l'évitement des obstacles. Si un détecteur du côté gauche détecte un obstacle, nous réduirons la vitesse du moteur gauche pour faire tourner le robot vers la droite, et inversement pour les détecteurs du côté droit.
5. **Apprentissage et Ajustement des Poids :** Bien que nous puissions débuter avec des poids fixes, l'application d'un apprentissage automatique, comme l'apprentissage par renforcement, pourrait être utilisée pour optimiser ces poids.
6. **Expérimentation et Ajustement :** Nous testerons le robot dans divers environnements pour évaluer son comportement d'évitement d'obstacles. En fonction des résultats, nous ajusterons les poids et la logique du réseau pour améliorer l'efficacité de l'évitement.

En résumé, en utilisant un réseau de neurones adapté aux spécificités des détecteurs booléens de l'Alphabot2, nous pouvons créer un système d'évitement d'obstacles efficace. Les ajustements des poids et la logique du réseau seront cruciaux pour développer un comportement d'évitement adaptatif et performant.

***Question 5*** *: Si vous deviez fournir un algorithme d’évitement d’obstacle pour l’Alphabot2, sous forme d’un automate à état fini, quel serait-il ( 1 ) ? Implémentez et testez.*

Pour élaborer un algorithme d'évitement d'obstacles pour l'Alphabot2 sous forme d'automate à états finis, nous avons défini plusieurs états clés et les transitions entre eux en fonction des entrées des capteurs. Notre objectif est de permettre à l'Alphabot2 de naviguer de manière autonome tout en évitant les obstacles rencontrés.

1. **États de l'Automate** :
   * **Avancer** : C'est l'état initial. Le robot avance en ligne droite.
   * **Éviter Gauche** : Le robot entre dans cet état lorsqu'un obstacle est détecté sur le côté gauche.
   * **Éviter Droit** : Cet état est activé lorsqu'un obstacle est détecté sur le côté droit.
   * **Arrêt** : L'Alphabot s'arrête. Cet état peut être utilisé dans des situations spécifiques, comme une intervention manuelle.
2. **Transitions entre les États** :
   * De **Avancer** à **Éviter Gauche/Droit** : Le robot change d'état lorsqu'un des capteurs détecte un obstacle.
   * De **Éviter Gauche/Droit** à **Avancer** : Si aucun obstacle n'est détecté, le robot retourne à l'état d'avancement.
   * Vers **Arrêt** : Le robot peut entrer dans cet état sous certaines conditions, comme un signal de l'utilisateur.
3. **Implémentation** :
   * Nous utilisons une boucle principale qui examine en continu les entrées des capteurs.
   * En fonction de ces entrées, l'état actuel du robot est mis à jour.
   * Des actions spécifiques sont exécutées dans chaque état (comme avancer, tourner, s'arrêter).
4. **Test de l'Automate** :
   * Nous testons le comportement de l'Alphabot2 dans un environnement avec des obstacles.
   * Nous observons et ajustons les paramètres pour optimiser l'évitement d'obstacles.

Ce processus nous permet de créer un système de navigation simple mais efficace pour l'Alphabot2, en utilisant des concepts de base de la robotique et de l'automatisation. Notre approche met en évidence la façon dont un automate à états finis peut être utilisé pour gérer des tâches complexes comme l'évitement d'obstacles dans la robotique.

***Question 6*** : Quel algorithme mettriez-vous en place pour un suivi de contours d’obstacles par la droite sur le Khepera III ? Implémentez et testez.

Pour développer un algorithme de suivi de contours d'obstacles par la droite pour le robot Khepera III, nous avons adopté une stratégie basée sur un ensemble d'états finis, similaire à celle utilisée dans l'évitement d'obstacles, mais avec une attention particulière au maintien d'une distance constante par rapport à l'obstacle.

* Détection d'Obstacle : Nous commençons par utiliser les capteurs de proximité latéraux du robot pour détecter la présence d'un obstacle sur son côté droit. Cela déclenche le début du processus de suivi de contour.
* Maintien de la Distance : Notre objectif principal est de maintenir une distance constante entre le robot et l'obstacle. Pour ce faire, le robot ajuste régulièrement sa trajectoire pour corriger toute déviation par rapport à cette distance cible.
* Navigation le Long de l'Obstacle : Le robot continue de suivre le contour de l'obstacle en gardant cette distance constante, ce qui lui permet de contourner efficacement l'obstacle sans s'en éloigner ni s'en rapprocher excessivement.

### **Implémentation de l'Algorithme**

Notre implémentation implique plusieurs états :

* AvancerLibrement : L'état par défaut, où le robot avance jusqu'à ce qu'un obstacle soit détecté.
* SuivreObstacle : Une fois l'obstacle détecté, le robot entre dans cet état pour suivre le contour de l'obstacle.
* AjusterDistance : Si le robot s'écarte de la distance souhaitée, il ajuste sa trajectoire pour revenir à cette distance.

Nous avons utilisé une boucle principale qui contrôle l'état actuel du robot et ajuste son comportement en fonction des lectures des capteurs.

### **Test de l'Algorithme**

Pour tester notre algorithme, nous avons placé le Khepera III dans un environnement avec divers obstacles et observé son comportement. Nous avons ajusté les paramètres, tels que la distance souhaitée par rapport à l'obstacle, pour optimiser le suivi de contour.

En résumé, cette méthode nous permet de réaliser une navigation précise et adaptative autour des obstacles, en exploitant les capacités des capteurs de proximité du Khepera III. Cet algorithme illustre comment des principes simples de robotique peuvent être utilisés pour accomplir des tâches de navigation complexes et dynamiques.

***Question 7*** : Si vous voulez implémenter un suivi de ligne grâce aux capteurs du robot Alphabot2, que proposeriez-vous comme algorithme ? Implémentez et testez.

Pour implémenter un système de suivi de ligne au sol avec le robot Alphabot2, nous avons développé un algorithme basé sur les données fournies par les capteurs du robot. Notre objectif est de permettre à l'Alphabot2 de suivre une ligne tracée au sol de manière précise et autonome. Voici comment nous avons abordé cette tâche :

* Utilisation des Capteurs de Ligne : Les capteurs au sol de l'Alphabot2 sont utilisés pour détecter la ligne. Ces capteurs peuvent identifier un contraste de couleur, tel qu'une ligne noire sur un sol blanc.
* Détermination de la Position par Rapport à la Ligne : En fonction des lectures des capteurs, nous déterminons si l'Alphabot2 se trouve directement sur la ligne, ou s'il dévie à gauche ou à droite.
* Ajustement de la Trajectoire : Selon la position actuelle par rapport à la ligne, nous ajustons la trajectoire de l'Alphabot2. Si le robot dévie à gauche de la ligne, il tourne légèrement à droite pour se recentrer, et inversement s'il dévie à droite.

***Question 8*** : Mettez en place deux stratégies de coordination différentes et testez les différences

Pour répondre à la question de mettre en place et de tester deux stratégies de coordination différentes pour un robot, nous pouvons choisir l'architecture de subsomption et l'architecture DAMN (Distributed Architecture for Mobile Navigation). Voici comment nous pourrions procéder pour chaque architecture et tester leurs différences :

### **1. Mise en Place de l'Architecture de Subsomption**

L'architecture de subsomption organise les comportements en couches, où chaque couche peut subsumer (c'est-à-dire, prendre le contrôle sur) les couches inférieures.

#### Implémentation

* Couches de Comportement : Nous définissons des couches de comportement telles que "éviter les obstacles", "suivre une ligne" et "explorer".
* Priorités : Les couches supérieures ont la priorité sur les couches inférieures. Par exemple, l'évitement d'obstacles peut prendre le pas sur le suivi de ligne.

### **2. Mise en Place de l'Architecture DAMN**

L'architecture DAMN utilise un système de vote pour coordonner les actions proposées par différents modules de comportement.

#### Implémentation

* Modules de Comportement : Créer plusieurs modules indépendants proposant des actions (par exemple, "éviter les obstacles", "suivre une ligne").
* Système de Vote : Chaque module attribue un score à ses actions proposées. Un module central recueille ces scores et sélectionne l'action ayant le score le plus élevé.

### **Test et Comparaison des Architectures**

Pour tester ces architectures, nous pourrions déployer un robot dans un environnement avec des obstacles, des lignes au sol et des zones d'intérêt.

#### Tests avec l'Architecture de Subsomption

* Observation : Voir comment le robot réagit lorsqu'il rencontre des obstacles tout en suivant une ligne.
* Attentes : Des transitions rapides et hiérarchiques entre les comportements, avec une priorité donnée aux couches supérieures.

#### Tests avec l'Architecture DAMN

* Observation : Évaluer la réaction du robot dans le même environnement.
* Attentes : Des actions plus nuancées, résultant d'un consensus entre les propositions des différents modules.

#### Comparaison des Résultats

* Réactivité : Comparer la rapidité et l'efficacité de la réaction aux changements environnementaux.
* Complexité du Comportement : Évaluer la capacité de chaque architecture à gérer des situations complexes.
* Flexibilité : Observer la capacité d'adaptation aux situations imprévues ou aux changements d'environnement.

En effectuant ces tests, nous pouvons comparer l'efficacité de chaque architecture en termes de gestion de comportements multiples et de leur capacité à répondre de manière appropriée et efficace à un environnement dynamique.