TR54
Modélisation et commande des systèmes temps-réel

Université de Technologie de Belfort Montbéliard

Croisement de trains de véhicules

Étudiant :
Nathan Olff
Felix Lahemade

8 janvier 2016

Table des matières

1. Introduction

Le projet consiste à gérer un croisement de trains de véhicules.

On définit un train (aussi nommé *platoon*) comme étant une entité virtuelle, constituée de plusieurs véhicules (2 ou plus) qui se suivent en gardant une certaine distance entre chacun d'entre eux.

L'objectif du projet est de faire rouler deux trains sur un même circuit devant se croiser et adapter leur vitesse afin d'éviter toute collision.

Ce projet est développé en collaboration avec un autre module : *Systèmes multi-agents et résolution distribuée de problèmes*. Dans le cadre de cette autre module, une simulation logiciel a été développé. Cette simulation possède quelques différences avec la version pratique étudié ici. Cependant, le problème principal reste le même : comment adapter la vitesse d'un train à l'approche d'un croisement pour éviter toute collision.

Une fois le travail sur la simulation terminée, une implémentation réelle sera effectuée à l'aide de robots Mindstorm EV3.

2. Simulation

2.1 Présentation

Le circuit utilisé en simulation est constitué de deux chemins fermés, chacun contenant un train de véhicules.

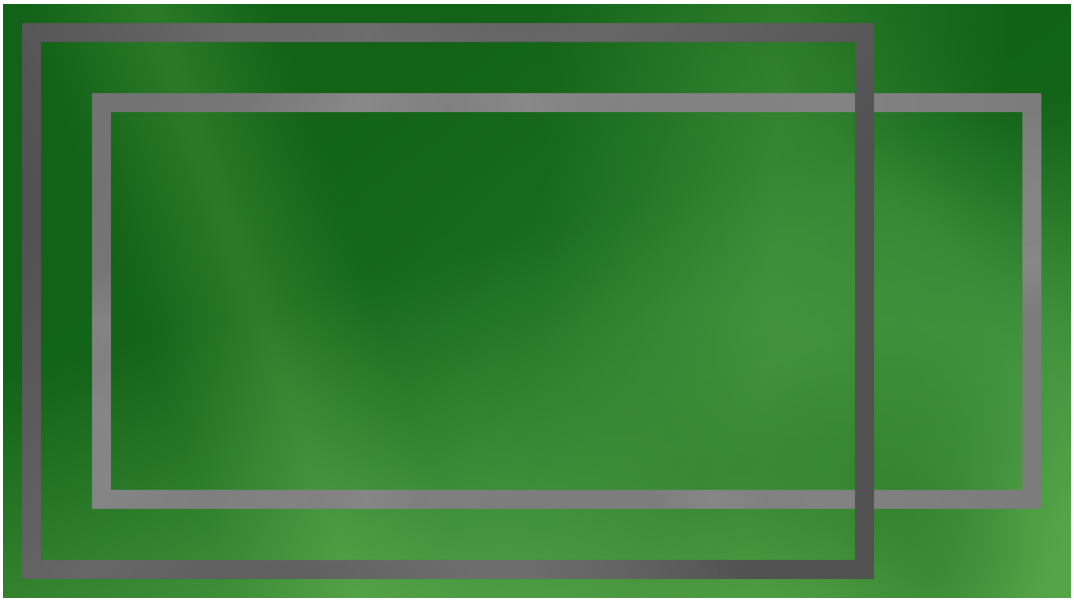

FIGURE 2.1 – Exemple de carte présentant les deux routes

Le but du projet est de permettre à deux trains de véhicules de traverser ces intersections sans avoir à s'arrêter ou à définir un ordre de priorité.

Comme présenté sur la figure 2.3, les véhicule du train A traversent le croisement entre deux véhicules du train B. Ce fonctionnement permet de ne pas réserver le croisement pour tel ou tel train, et produit donc un trafic plus fluide qu'une solution type feu rouge.

FIGURE 2.2 – Train formé de quatre véhicules

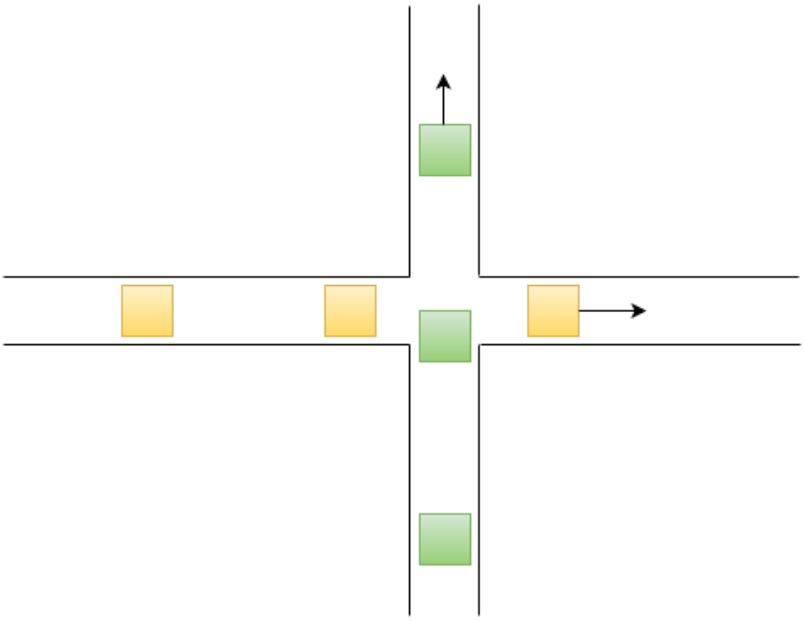

FIGURE 2.3 – Croisement de trains

Afin d'atteindre ce but, les trains doivent adapter leurs vitesses et leurs inter-distances à l'approche d'un croisement.

2.2 Implémentation

Le développement de cette simulation utilise les principes de systèmes multi-agents. Chaque véhicules et chaque train est modélisé par un agent.

La plateforme multi-agent MadKit ainsi que le framework graphique Swing ont été utilisé.

2.2.1 Définition des agents

Afin de modéliser le problème de croisement de trains de véhicules, j'ai utilisé deux types d'agents : véhicule (appelé *Car*) et train, chacun représenté par un classe héritant de la classe MadKit Agent.

Agent véhicule

Chaque véhicule est représenté par un agent Car. Ses attributs sont répartis en plusieurs catégories :

— Identité : l'identifiant unique de l'agent, l'identifiant de son train ainsi que sa position à l'intérieur du train

— Position : la position absolue du véhicule sur la carte, mais également la liste des intersections dans lesquels le véhicule se trouve [1]

— Mouvement en cours : la vitesse actuelle ainsi que la distance entre lui et le véhicule précédant. Il existe d'autres attributs, utilisé pour le calcul de vitesse du premier véhicule du train. Ces derniers seront détaillés dans la partie concernant le comportement du dit *Leader* (voir section 2.2.3).

— Mouvement optimum : vitesse et distance du précédant à atteindre. Afin d'avoir une certaine fluidité de trafic, l'accélération et décélération des véhicules sont bornées. Cela peut donc entraîner une différence entre la vitesse courante et l'objectif de vitesse.

Agent Train

L'agent Train est un agent virtuel. Nous verrons dans la section 3 que leurs comportement a été intégré à celui du Leader dans la pratique.

Le train possède également plusieurs types d'attributs :

— Identité : numéro du train

— Position : liste de croisements dans lesquels au moins un véhicule du train se trouve. Une seconde liste répertorie les croisements dans lesquels le train ainsi qu'un autre train sont présents (gestion de conflit, voir section 2.2.3).

— Objectifs : vitesse moyenne du train et objectif d'inter-distance à maintenir

2.2.2 Passage d'un train dans une intersection

Voici un diagramme de séquence présentant les principes utilisés, ainsi qu'une description de chacun des niveaux de communications.

1. Un véhicule est considéré à l'intérieur d'une intersection lorsqu'il est à moins d'une certaine distance de la dite intersection. Si deux intersections sont trop proches, il pourrait donc arriver qu'un véhicule soit dans deux intersections à la fois. Ce cas de figure n'ayant pas été étudié, nous n'utiliserons pas ce type de cartes.

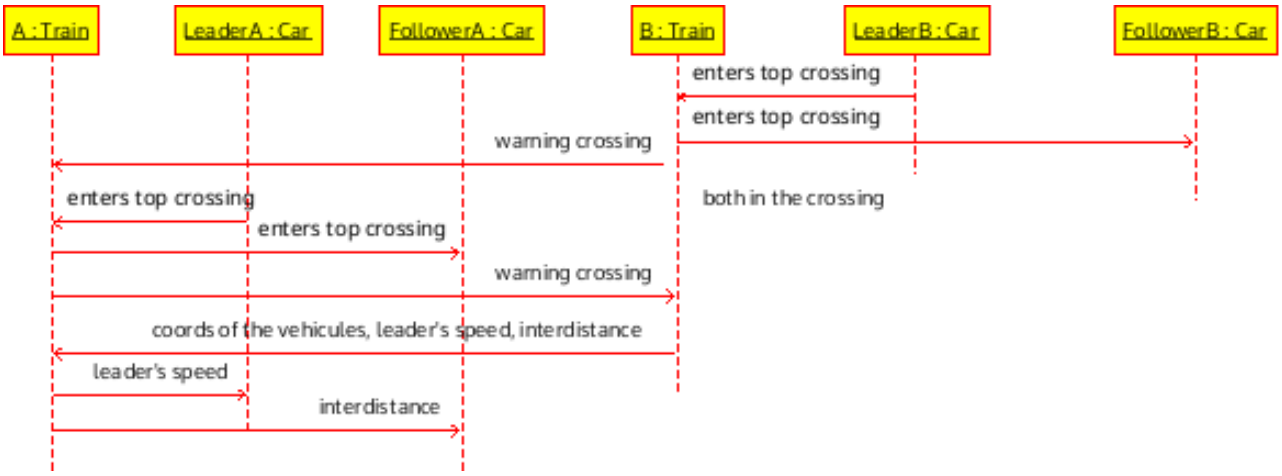

FIGURE 2.4 – Diagramme de séquence – Arrivé des deux trains dans un même croisement

Véhicule → Train : Remontée d'informations des véhicules vers leur train

Le premier véhicule (aussi appelé *Leader*) de chaque train a un rôle particulier à jouer. Il est le seul à appliquer strictement les ordres de vitesses données par son train (voir section 2.2.3). Il a aussi pour rôle de prévenir le train, via l'envoie d'un message, lorsqu'il arrive à proximité d'une intersection.

La queue du train (aussi appelé *Tail*) se charge, dès lors qu'elle a quitté une intersection, de prévenir son train de cette même sortie.

Train → Véhicules : Transmission de l'information vers les véhicules du train

Lorsque le train reçoit un message d'entrée ou de sortie de croisement, il transfère un message de même type à tous les éléments le composant. Ainsi, chaque voiture du train sait qu'il se trouve à proximité (ou non) d'un croisement et peut donc agir en conséquence.

Train → Trains : Prévenir les autres trains d'une entrée ou sortie de croisement

Lorsqu'un train rentre ou sort d'une intersection, il envoie un message d'avertissement aux autres trains [2] pour le(s) prévenir de sa position.

2. Seul les systèmes avec deux circuits, et donc deux trains au total ont été étudié ici.

2.2.3 Comportements des agents

Dans la simulation, les agents sont exécutés en parallèle, chacun dans une boucle contenant son exécution ainsi qu'une attente correspondante au pas de la simulation (fixé pour les essais à 35ms).

Agent véhicule

A chaque itération, l'agent véhicule effectue les actions suivantes :

1. Réception des messages.
2. Modification de l'état interne de l'agent en fonction des messages reçus.
3. Définition d'un objectif de vitesse suivant les messages précédent et le distance au précédant.
4. Exécution de l'ordre de vitesse dans la limite de l'accélération et décélération maximales.

Comme on peut le voir ci-dessus, les agents véhicules se contentent donc d'agir en fonction de l'observation qu'ils ont de l'environnement et des messages reçus par leur train. Il n'y a donc ni planification ni processus de réflexion. Ces agents sont bien des agents de type réactifs.

Tête du train

La tête du train, également appelé *Leader*, a pour rôle d'appliquer les ordres de vitesses envoyé par son train. La plupart du temps, il s'agit d'une vitesse constante à maintenir.

Il peut arriver que le train n'envoie pas une vitesse constante, mais une fonction polynomiale définissant la courbe de vitesse à suivre jusqu'à atteindre le croisement. Dans ce cas, le leader calcule à chaque itération la valeur de la fonction en fonction du temps écoulé depuis que le polynôme a été défini. (voir section 2.2.3)

Comportement des suiveurs

Les autres véhicules du train sont des suiveurs. Ils se contentent d'appliquer le dernier ordre d'inter-distance transmis par leur train.

Le respect de cet inter-distance est défini en utilisant un régulateur PID (Proportionnel, Intégral, Dérivé).

Le régulateur PID consiste en une fonction ayant pour but de diminuer une erreur par rapport à une valeur idéal. Dans notre cas, l'erreur est calculée en prenant la différence entre l'inter-distance idéale et la distance d'avec le véhicule précédant. Il prend en compte l'historique d'erreur (grâce à la partie intégrale de la fonction), une partie prédiction de l'erreur à venir (partie dérivée), ainsi qu'une partie proportionnelle à l'erreur courante.

Les paramètres (Ki, Kd, Kp) du PID ont été définit suite à plusieurs essais.

Queue du train

La queue du train (appelé *Tail*) est le dernier véhicule du train. Son comportement est celui d'un suiveur, à la différence près que lorsqu'il sort d'une intersection, il envoie un message à son train pour l'en notifier.

Agent train

Le train étant un agent virtuel, il n'a pour but que de recevoir des messages, de définir un comportement global, et de transmettre ce comportement sous forme d'ordres à ses véhicules. Il fait le pont entre les agents véhicules qui le composent et l'autre train.

Lorsqu'un train A est dans une intersection et reçoit un avertissement indiquant l'entrée d'un autre train B dans cette même intersection, il lui répond en envoyant les coordonnées de chacun de ses véhicules.

Le train B étudie alors les positions et vitesses des véhicules du train A pour définir une stratégie de croisement. À partir de la position et de la vitesse de son propre leader, il calcule combien de temps il va mettre (à vitesse actuelle) pour atteindre le croisement. Il calcule également le temps que va mettre chacun des véhicules du train A pour atteindre le croisement.

Avec ces deux informations, le train B recherche entre quels deux véhicules du train A son leader devrait passer. Il calcule par la suite une vitesse moyenne optimale légèrement différente de la vitesse actuelle afin de permettre au leader d'atteindre le croisement au milieu des véhicules de A.

Dans un premier temps , nous avons essayé de transmettre cette nouvelle vitesse moyenne aux membres du train. Le leader atteignait alors bien le croisement au moment définit, mais les suiveurs étaient de plus en plus éloigné du comportement "idéal".

En réfléchissant un peu, on peut voir qu'il y avait effectivement un problème de conception dans cette solution.

En effet, pour que les véhicules des deux trains puissent se croiser correctement une fois que le leader a atteint le croisement, il faut que les inter-distances et les vitesses des deux trains soient identiques.

A partir de ce constat , nous sommes partis à la recherche d'une solution alternative, respectant ce besoin de même vitesse au niveau du croisement.

Afin d'obtenir une solution viable, il fallait trouver une fonction de vitesse en fonction du temps respectant les données ci-dessous :

— La vitesse moyenne calculée précédemment doit être respecté (calcule de valeur moyenne grâce à l'intégrale)
— La vitesse de départ de la fonction est connue.

Algorithm 1 Comportement d'un train lorsqu'il reçoit les coordonnées des véhicules d'un autre train

```
tempsLeaderCroisement ← distance(leader, croisement)/vitesseLeader
Pour tout véhicule v du train opposé Faire
  tempsRestant[v] ← distance(v, croisement)/vitesseDuTrainOpposé
Fin Pour
minTemps ← 0
maxTemps ← infini
Pour tout valeur t de tempsRestant Faire
  Si t < tempsLeaderCroisement Et t > minTemps Alors
    minTemps ← t
  Sinon Si t >= tempsLeaderCroisement Et t < maxTemps Alors
    maxTemps ← t
  Fin Si
Fin Pour
Si minTemps = 0 Alors
  minTemps ← maxTemps - interdistanceDuTrainOpposé/vitesseDuTrainOpposé
Fin Si
Si maxTemps = infini Alors
  maxTemps ← minTemps + interdistanceDuTrainOpposé/vitesseDuTrainOpposé
Fin Si
minTemps ← minTemps + tailleVoiture/vitesseAutreTrain
maxTemps ← maxTemps - tailleVoiture/vitesseAutreTrain
tempsOptimal ← (minTemps + maxTemps)/2
vitesseOptimal ← distanceLeaderCroisement/tempsOptimal
Si |vitesseOptimal - vitesseDuTrainOpposé| < 5 Alors
  vitessePolynome ← vitesseOptimal
Sinon
  vitessePolynome ← calculePolynome(vitesseLeader, vitesseDuTrainOpposé, distanceLeader-
  Croisement, tempsOptimal)
Fin Si
envoieMessage(véhicules, vitessePolynome)
```

— La vitesse d'arrivée, à t = durée optimal calculée précédemment, est égale à la vitesse de l'autre train.

— L'accélération et la décélération doit être les plus faibles possible. Il faut donc éviter toute fonction contenant des piques.

Polynôme de vitesse

Une solution réalisable relativement facilement consiste à représenter la vitesse sous la forme d'un polynôme de second dégrée.

La vitesse serait donc présenté sous la forme $at^2 + bt + c$ en fonction du temps. c est définie rapidement grâce à la vitesse initiale. Concernant les coefficients a et b, ils sont définie grâce à la résolution d'un système de deux équations à deux inconnus défini ci-dessous.

A $t = 0$, on a : $at^2 + bt + c = c = vitesseLeader$

Au croisement, on a $t = t_f$ = durée optimal pour atteindre le croisement

$at_f^2 + bt_f + vitesseLeader = vitesseDuTrainOppose$ (1)

En intégrant le polynôme de départ, on obtient :

$a\frac{t_f^3}{3} + b\frac{t_f^2}{2} + vitesseLeader \times t_f = distanceLeaderCroisement$ (2)

La résolution du système d'équation de (1) et (2) permet d'obtenir les coefficients a et b du polynôme de vitesse.

2.2.4 Résultats

Une fois la solution du polynôme implémentée, une série de tests a été effectuée pour vérifier l'efficacité du la méthode sur le long terme. En termes de paramétrage de la solution, il semble qu'une vitesse d'environ 80 pixels par secondes et une inter-distance égale à 6 fois la taille d'un véhicule donne de bons résultats.

Les résultats obtenus en simulation étaient donc très prometteurs. Malheureusement, l'implémentation sur les robots nous n'a pas donné les même résultats comme nous allons le voir dans la suite de ce rapport.

On voit bien sur la figure 3.1 l'application du polynôme de vitesse au Leader du train 1, ainsi que l'application, après un temps de propagation, de polynômes plus "aplatis" sur les véhicules suiveurs.

Cette différence s'explique par le fait que les suiveurs n'appliquent leur vitesse qu'en réaction à ce qu'il se passe devant eux. Ainsi, au moment où le Leader atteint le sommet de sa courbe de vitesse et commence à ralentir, le deuxième véhicule perçoit une différence d'inter-distance plus faible et commence donc lui aussi à ralentir.

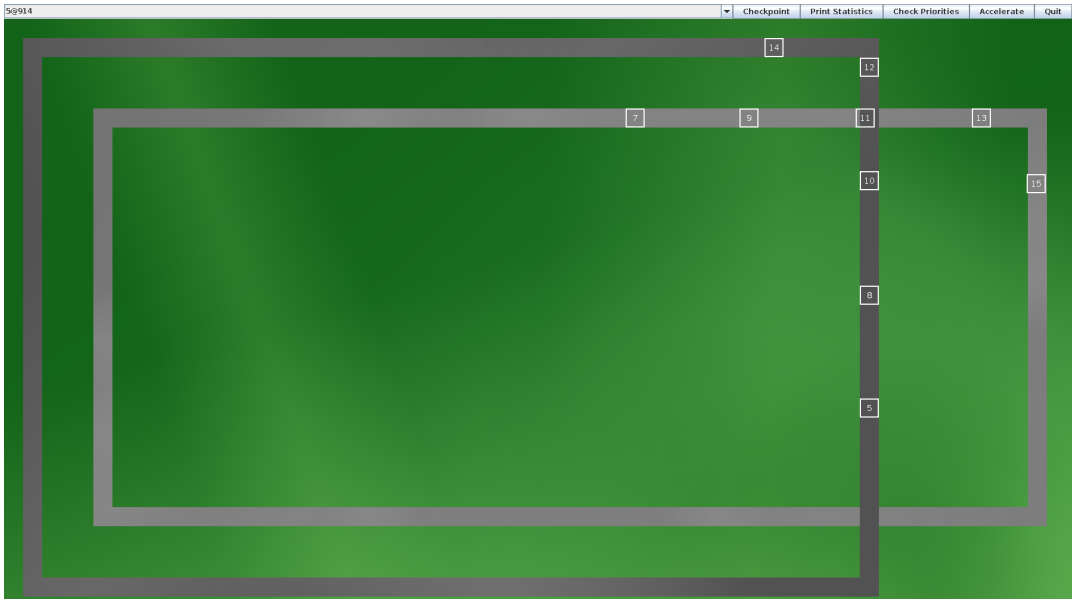

FIGURE 2.5 – Exemple de croisement

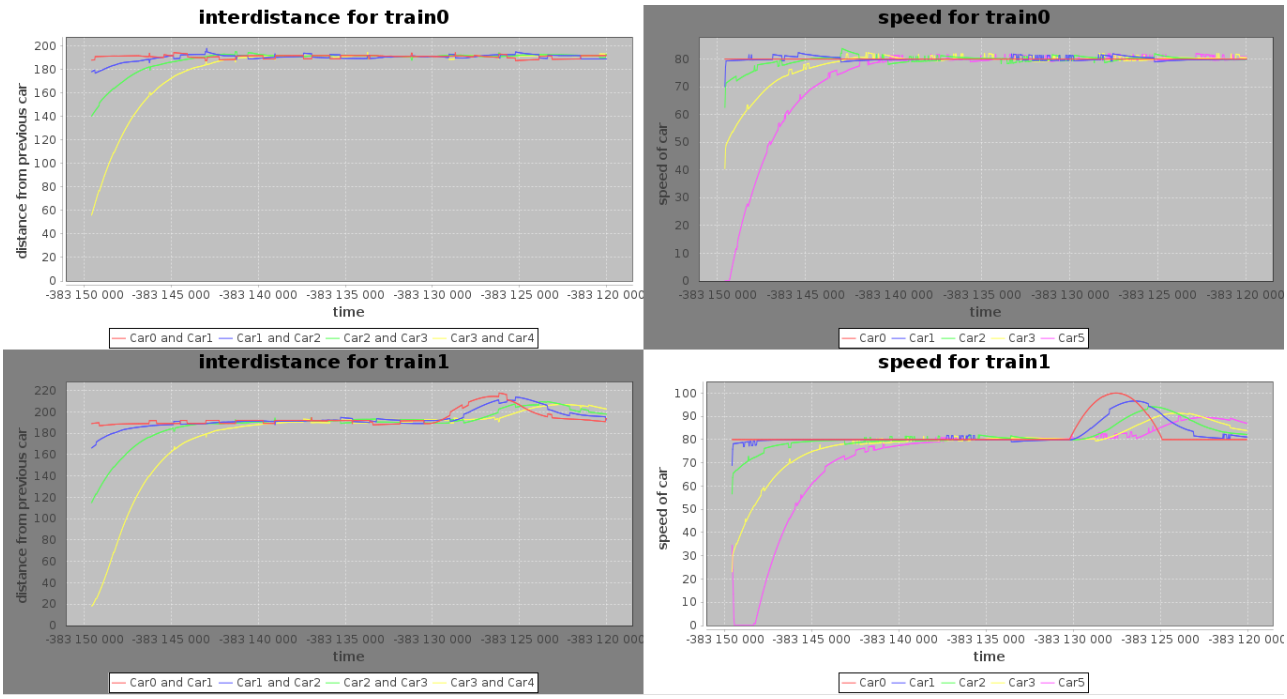

FIGURE 2.6 – Statistiques liés au croisement ci-dessus

3. Pratique

Pour le projet de TR54, nous avons tenté d'implémenter le même solution utilisée en simulation. Nous allons voir que les résultats obtenus sont loin d'être optimaux.

3.1 Différence de circuit

En pratique, les deux trains de véhicules circulent sur un seul circuit sous forme de huit.

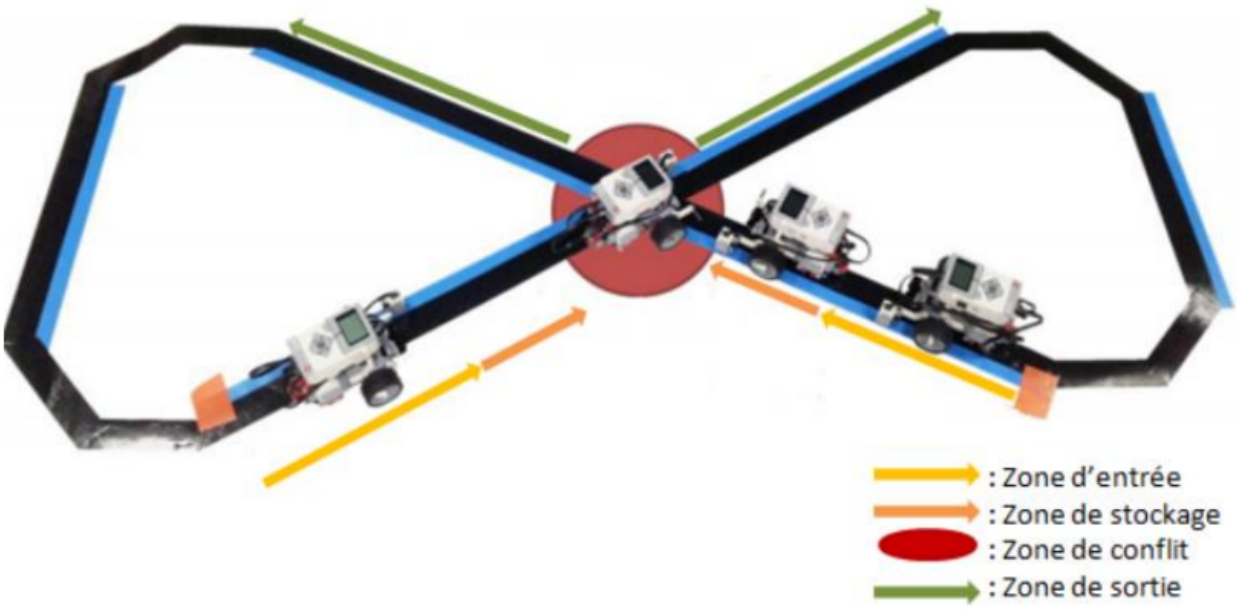

FIGURE 3.1 – Circuit physique

3.2 Structure principal du programme

Au démarrage, chaque robot propose un menu permettant à l'utilisateur de définir quel est le rôle de ce robot (Leader, Milieu ou Queue du train) ainsi que le numéro du train.

Une fois le choix effectué, le programme instancie la classe correspondante à son rôle et exécute ensuite sa fonction `live()`.

Les comportements de chaque type de robot est très proche de ceux utilisés par les agents de la simulation :

1. Réception des message
2. Modification de l'état interne en fonction des message
3. Détection de la couleur grâce au capteur.
4. Adaptateur de la vitesse individuel de chaque moteur en fonction de la couleur pour le suivi de ligne

3.3 Implémentation des agents trains

Les agents trains définis dans la simulation comme des agents à part entière, sont ici implémentés à l'intérieur même des voitures Leader.

3.4 Communication

Les messages sont envoyés sous forme de `DatagramPacket` via UDP en Wi-fi. Tout les messages sont transmis en broadcast à tout les robots présents sur le réseau.

Chaque robot possède un identifiant unique défini par son rôle et son numéro de train : $carId = NbTrain * 10 + role$ (avec Leader = 1, Middle = 2, Queue = 3). L'identifiant 21 correspond par exemple au Leader du train numéro 2.

La classe `MessageDatagram` est utilisée pour contenir le message (objet de la classe `Message`) à transmettre ainsi que des métadonnées le concernant :

— l'id du robot émetteur
— l'id récepteur
— le timestamp du robot émetteur

Ce `MessageDatagram` est sérialisé et envoyé dans un `DatagramPacket` à tout les robots. À la réception, chaque robot décode le `MessageDatagram` et compare l'id récepteur à son propre identifiant. Pour envoyer un message à tout les véhicule d'un même train, il suffit d'utiliser le numéro du train multiplié par 10 comme identifiant récepteur.

En UDP, il est possible que des paquets soient corrompus ou perdus. Afin d'éviter les désagrément dû à la disparition d'un paquet, nous avons décidé d'inclure le timestamp du robot émetteur dans le `MessageDatagram`. Chaque robot maintient une map contenant les timestamps du dernier message reçu depuis un identifiant précis. Cela nous permet donc à chaque envoi de message, d'envoyer le paquet plusieurs fois sans que les robots n'effectuent plusieurs fois l'action à la réception.

La réception des paquets est effectuée dans un second thread. Ce thread d'écoute se contente de recevoir les paquets, de détecter s'il lui est destiné, et s'il l'a déjà reçu (grâce au timestamp). Une fois filtré, les `Message` sont extraits des `MessageDatagram` et ajoutés à une file FIFO à accès concurrent appelé `mailbox`.

Le thread principal se contente donc juste, à chaque tour de boucle, de lire tous les messages contenus dans la `mailbox`.

3.5 Coordonnées de véhicules

Dans la simulation, chaque robot fonctionne par itération. Toutes les 35ms, un pas de simulation est effectué. Lors de cette itération, le robot avance d'un certain nombre de pixels sur un chemin défini en dur dans le programme. Il est alors très facile, à partir de la vitesse, de calculer la position du robot à chaque itération.

En réalité, les robots ne sont pas conscients de leur position. Afin de définir une position approximative du robot sur le circuit, on utilise des marques oranges présentes sur le circuit, ainsi que le nombre de tour de roue effectué par les robots. Le nombre de tour de roue est défini grâce aux méthodes `motor.getTachoCount()`. À chaque marque orange, on remet à zéro la valeur du tacho count utilisée.

Nous avons remarqué de légères différences de comportement sur les robots. Il est possible que les capteurs ne soient pas tous calibrés de la même façon. On se retrouve donc avec certains robots qui avancent parfaitement droit sur les lignes bleus, et d'autres qui ne détectent pas le bleu et se déplacent en zig zag. Ces différences de comportement engendrent une grande incertitude sur les estimations de position. En effet, un robot circulant droit atteindra le croisement en environ 4 tours de roues, alors qu'un robot se déplaçant en zig zag sera plus lent et mettra environ 5 à 6 tours de roues.

3.6 Suivi de véhicule

Dans la simulation, le respect de distance entre chaque véhicules est relativement simple. En pratique, le cône de détection du capteur de distance est relativement faible, ce qui empêche tout détection au niveau des virages.

Un simple suivi avec politique "à un point" est effectué pour le maintien de l'interdistance, et non un contrôleur PID comme en simulation. L'utilisation du PID aurait engendré une complexité accrue, notamment dûe à l'obligation de définir ses paramètres Kp, Kd et Ki.

De plus, le maintien de l'interdistance grâce au capteur de distance entre en conflit avec notre principe de simulation. Ainsi, les robots sont censés maintenir une interdistance constante avec les autres membres de leur trains, mais doivent frôler ceux de l'autre train. Cependant, le capteur ne permet pas de faire la différence entre un robot de son propre train, et un robot de l'autre train.

3.7 Suivi de ligne

Le capteur de couleur nous renvoie un tableau de trois réels. Nous effectuons alors plusieurs tests sur les 3 valeurs de ce tableau afin de définir la couleur se situant sous le capteur : noir, bleu, blanc ou orange (clair ou foncé).

En fonction de la couleur détectée, on définit une politique de variation de vitesse entre les deux moteurs :

— Noir : Lorsque le comportement lié au noir est appliqué (soit avec la couleur noir, soit avec l'orange foncé), on fait tourner le moteur gauche à la moitié de la vitesse du moteur droit. Le robot tourne donc légèrement vers la gauche.
— Blanc : Lorsque le comportement lié au blanc est appliqué (soit avec la couleur blanche, soit avec l'orange clair), on fait tourner le moteur droit à la moitié de la vitesse du moteur gauche. Le robot tourne donc légèrement vers la gauche.
— Bleue : Lorsque la couleur bleue est détectée, on fait tourner les deux moteurs à la même vitesse.

La mise en commun de ces trois comportements résulte en un robot qui suit la limite gauche du scotch noir, et qui avance droit lorsqu'une bande bleue est détectée.

3.8 Résultats

Nous avons effectué de nombreux tests sur le programme final. Les résultats se trouvent être en demi-teinte. Les messages sont bien transmis et arrivent à destination. Le train reçoit les messages d'entrée et de sortie des croisement. Malheureusement, les collisions ne sont pas évitées.

N'ayant pas beaucoup d'expérience avec la robotiques, nous ne pouvons pas identifier avec certitude les causes exactes des problèmes rencontrés. Nous pensons cependant que les deux faits suivants ont eu un effet sur les résultats obtenus :

— Les incertitudes sur les positions s'additionnent lors du calcul d'évitement de collision. Dans la simulation, les robots se frôlent presque à chaque fois, sans toutefois se toucher. Dans la réalité, on comprend vite qu'avec les incertitudes de bases, les résultats des calculs ne peuvent être corrects.
— Le temps de traitement et de transfert par wi-fi implique qu'au moment où le train commence à traiter le conflit à venir, tout les robots ont déjà avancé d'environ 10 à 15 cm. De ce fait, même si les estimations étaient parfaites, elles seraient appliquées en retard.

4. Améliorations possibles

Plusieurs améliorations pourraient être apportés au projet :

Si nous avions la possibilité de calibrer les capteurs de couleur, peut-être pourrions-nous éviter les différences de comportements observées (et l'absence de détection du bleu pour certains robots).

Un circuit plus grand aurait probablement été plus approprié à l'implémentation de notre solution. En effet, avec une zone de préparation plus grande entre les marques orange et le croisement, les véhicules auraient peut-être plus de temps pour adapter leur vitesse et suivre les ordres envoyés par le train. Lorsque l'on regarde la simulation, on voit que la proportion entre le circuit et les véhicules est loin d'être la même que celle entre le circuit en huit et les robots EV3.

L'utilisation d'un circuit plus grand nous permettrait également de proposer des virages moins serrés, résolvant le problème du capteur de distance dans les virages. Une autre solution pour régler ce problème serait l'utilisation de capteurs ayant une zone de détection plus large.

Nous pourrions également essayer de baisser la vitesse générale de tous les robots afin de réduire l'impact du temps de traitement et de transfère sur le résultat.

En ajoutant des marqueurs sur chaque robots ainsi qu'une caméra avec reconnaissance d'image, on pourrait obtenir des coordonnées de véhicules beaucoup plus précis. La détection à l'aide de caméra mériterait cependant un projet à lui seul.

5. Conclusion

Le croisement de trains de véhicules est un sujet très intéressant, notamment lorsque l'on réfléchit aux applications dans le domaine des véhicules autonomes et aux questions de gestion de trafic que ces derniers vont générer.

Le travail préalable effectué pour le développement de la simulation nous a permis de valider notre modèle de gestion des conflits et présente des résultats intéressants.

Nous nous doutions dès le départ que l'implémentation d'une solution viable pour un problème de ce type allait présenter des difficultés lors du passage à la pratique, cependant, le challenge qui en découlait était vraiment intéressant et nous avons éssayé jusqu'au bout de résoudre les différents problèmes successifs pour arriver à la version actuelle du développement.

Malgré les résultats en demi-teinte de notre programme, son développement a été une activité très intéressante. La partie communication fonctionne correctement, le suivi de ligne fonctionne également de manière satisfaisante ainsi que l'entrée et la sortie des trains du croisement.

Nous sommes conscients d'avoir été ambitieux dans les objectifs de ce projet et de n'avoir pas parfaitement atteint l'ensemble de ces objectifs. Cependant nous avons le sentiment d'avoir produit un projet abouti auquel il ne manquerait pas grand chose pour qu'il remplisse complètement ces objectifs.