

LABORATOIRE DE SYSTÈMES ROBOTIQUES

Thymio Road Network



Benjamin KERN

Professor:
Fransesco MONDADA

Assistants:
Christophe BARRAUD
Stefan WITWICKI

June 1, 2015

SEMESTER PROJECT — SPRING 2014–2015			
Title:	Thymio Traffic Network - Global Infrastructure		
Student:	Benjamin Kern	Section:	Électrique/Électronique
Professor:	Francesco Mondada		
Assistant 1:	Stefan Witwicki	Assistant 2:	Christophe Barraud

Project Statement

Context. Thymio is a small mobile robot developed here at EPFL for education and for fun. It has two wheels, 9 infrared sensors, an accelerometer, a microphone, speakers, and 39 brightly-colored LEDs. Imagine a community of Thymios coexisting in an urban environment. Each Thymio has a purpose; and with that purpose, things to do and places to go. Here, our motivation is to build up a transportation infrastructure for the Thymio society in the form of a road network.

The road network will be realized in two parts : *local* infrastructure for a Thymio's movement with the network, and *global* infrastructure, for the functioning of the network as a whole. This project is focused on the latter part, though will be performed in close collaboration with other half of the project.

Core Objectives.

- Design of a city. The student should consider how to achieve fluid flow of traffic of the robots. The city should include a certain list of special places (train station, police building...). It should include different sorts of lanes (slow, fast, one-way, two-way, etc.)
- Design of intersections (in collaboration with the student building the *local infrastructure*). The choice of an appropriate intersection method should be made (crossroad, roundabout, traffic lights, traffic barrier, etc.). The expected advantages and disadvantages of all options should be weighed, and one option should be carefully selected.
- Manage interactions in an intersection. The robots should be able to follow the rules of the intersections and avoid collisions.
- High-level decision making strategies. To allow Thymios to decide which pathways to take, the student should explore at least two different strategies (random, path planning (point A to B), predefined route, coordination?) He should analyze these strategies and discuss them.

Additional Objectives. The following constitute additional goals whose treatment can each significantly increase the contribution of the project *assuming* the core objectives have already been fulfilled to a reasonable degree of satisfaction.

- Develop a coordinated joint control strategy for the Thymios, by which they can cooperate to make the network more efficient (could be done in collaboration with the other student).
- Implement an internal map representation for one or more Thymios to use for more advanced decision-making.
- Create high priority vehicles (police car, ambulance, street cleaner...) and make them have the priority on regular robots (could be done in collaboration with the other student).
- Your own creative ideas (to be discussed with the project assistants).

Observations

A project schedule will be established and submitted to the assistants in charge before **the 2nd of March, 2015**.

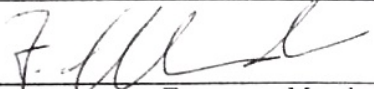

An intermediate presentation (consisting of approximately 10 minutes of presentation and 10 minutes of questions and answers) of your work will be held **during the first weeks of the month of April**. The goal here is to give a quick overview of the work already accomplished, to propose a precise plan for the remainder of the project, and to discuss the various options thereof.

A report, including as a preamble the project statement (this document), and followed by a one-page abstract (according to the official canvas), will be submitted **on Monday, 1st of June 2015, before 12:00**, to the project assistant in 3 copies. The focus will be on the experiments and the obtained results. The target audience is an engineer (e.g., EPFL bachelor or master) without extended knowledge of the domain. A preliminary version of the report will be submitted to the assistant(s) on **the 22nd of May 2015**. All documents in digital version including the report (source and pdf), the document used for the oral presentation, an abstract in pdf format, as well as the source code of any developed programs, must be submitted to the assistant according to the instructions you will receive.

A 30-minute defense of the project (approximately 20 minutes of presentation and demonstration, and 10 minutes of questions and answers) will be held **between the 1st and 12th of June 2015**.

Professor in charge:

Assistant in charge:

Signature: 	Signature: 
Francesco Mondada	Stefan Witwicki

Lausanne, the 17th of February 2015

Thymio road network

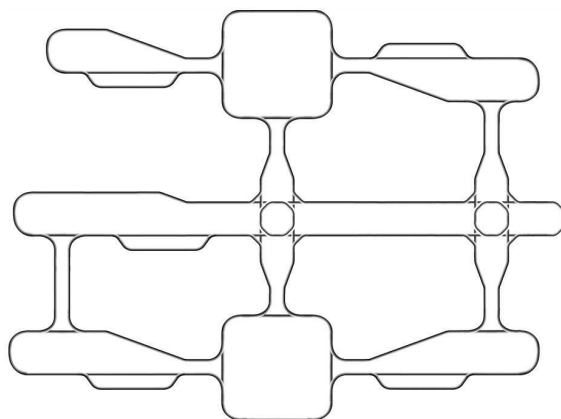
Benjamin Kern, El

Assistant 1: Christophe Barraud

Assistant 2 : Stefan Witwicki

Professeur: Francesco Mondada

Dans ce projet, nous avons créé un réseau de transport dans lequel une quarantaine de Thymio peuvent se déplacer librement et simultanément d'un endroit à un autre. Ils utilisent des lignes en dégradé de gris pour se diriger et disposent de parking pour se reposer. La carte est imprimée sur ensemble de posters d'une taille de 5 sur 5 [m]

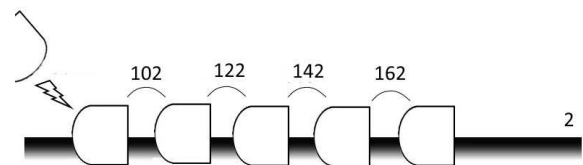


Le réseau de transport

Les Thymio reçoivent des informations principalement par la lecture de codes-barres qui sont placés le long des routes. En raison de la petitesse de ces codes (Pour des questions de fiabilités), les Thymio utilisent une carte interne pour interpréter leur position en fonction de leur parcours.

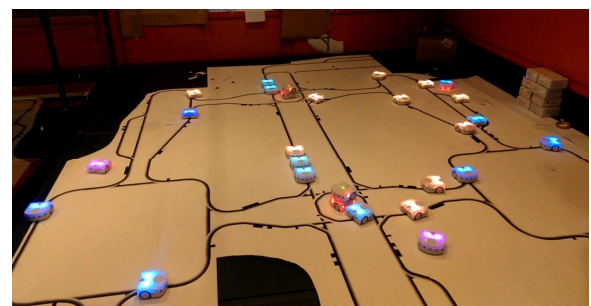
Les parcours sont ponctués de multiples intersections variées. Pour la gestion des carrefours, les robots possèdent aussi des protocoles de communications pour suivre les ordres des robots spécialisés qui gèrent les intersections et évitent les collisions.

La communication et le suivi de lignes sont gérés par capteurs infrarouges ce qui les rend sensibles à la lumière et donc difficile à gérer.



Déplacement d'un ping qui sert à mesurer la quantité de Thymio arrêté sur une ligne.

Les robots signaux peuvent de plus récupérer des informations sur la position pour adapter les priorités de trafic.



Carte finale en phase de test

Contents

1	Introduction	1
2	Description du projet	2
2.1	Le robot	2
2.2	Environnement de programmation	2
2.3	Objectifs et division du projet	2
3	Réseau minimal fonctionnel	3
3.1	Fonctions nécessaires	3
3.2	Routes	3
3.3	Les intersections	4
3.4	Parking	5
3.5	Technique d'information sur les lieux et les routes	6
3.6	Comportement des Thymio	6
3.7	Gestion des collisions	7
3.8	Prototype, tests et analyse	7
3.9	Conclusion de la section 3	7
4	Les complexifications et améliorations	9
4.1	Gérer les intersections et la coordination	9
4.2	Raisons du développement de la carte interne	9
4.3	Fonctionnalités et réalisation de la carte interne	10
4.4	Communication	14
4.5	Les protocoles de communication inter-thymio	15
5	Contrôle et mesures du système	18
5.1	Théorie et modélisation de réseaux routier	19
5.2	Contrôle des parking	19
5.3	Contrôle des intersections	19
5.4	Programmation et implémentation	21
5.5	Tests	21
5.6	Conclusion de la partie contrôle	22
6	Possibilités futurs	22
7	Conclusion	23
	Appendices	I
A	Tableaux de codes	I
B	Code Thymio voiture	II
C	Code Thymio signaux	IX
D	Code signal de parking	XII

1 Introduction

Le Thymio est un petit robot utilisé pour l'éducation aux technologies et à la programmation ainsi que pour les loisirs. C'est un robot abordable et robuste conçu pour supporter des conditions d'utilisation difficiles.

Dans ce projet, nous avons créé un réseau de transport dans lequel une quarantaine de robots peuvent se déplacer librement d'un endroit à un autre.

Pour le réaliser nous avons dû construire et assembler toutes les parties nécessaires à la conception d'un réseau routier adapté au format du Thymio.

Le Thymio possède de base de nombreux capteurs qui sont suffisants pour réaliser un système efficient et varié. Les capacités computationnelles sont elles, mobilisées à leur maximum.

En raison de la quantité de travail que représente le projet, celui-ci a été divisé en deux grandes parties. La première, la partie locale, se concentre sur le comportement bas-niveau des robots, c'est à dire sur la création des infrastructures et le comportement programmé des robots en relation à celles-ci. La deuxième, la partie globale, comporte tous les développements relatifs au système complet et contient toutes les parties nécessaires au fonctionnement coordonné des Thymios. Cela comprend les choix conceptuels des infrastructures de base jusqu'au développement des outils de gestion des flux. La première est réalisée par Loïc Dubois et la seconde par moi-même. Ce rapport se concentre donc sur la partie globale du réseau.

Pour le réaliser, nous avons procédé en 3 grandes étapes chronologiques. **La première étape** porte sur le développement d'un réseau fonctionnel minimal. Un réseau qui est capable de fonctionner avec une carte simple et qui utilise le minimum possible des composants vitaux d'un réseau routier. Ces éléments basiques sont des routes, un type d'intersection, une possibilité de parking, un moyen de lecture d'informations mais aussi un moyen d'éviter les collisions et la programmation des Thymios voitures afin de permettre l'utilisation simultanée du réseau par plusieurs Thymios. En commun avec M. Dubois, nous avons choisi des routes sous forme de lignes en dégradé de gris, des ronds-points pour les intersections, des voies de gares pour les parkings, des codes bars pour les différentes informations (limites de vitesses, cédez le passage, etc). Les collisions, elles sont contrôlées par les capteurs de proximité. Une fois ces éléments définis et conceptualisés, nous avons fait une carte qui les comporte tous.

La deuxième étape, s'est faite plus d'une façon individuelle car une fois ce réseau simple réalisé, chacun a pu se concentrer sur ses objectifs respectifs. Loïc a continué d'améliorer les comportements bas-niveau et rajouté différentes intersections, et moi à développer des outils pour gérer le système d'une manière coordonnée. Le but étant de pouvoir contrôler correctement le flux du trafic dans le circuit et d'améliorer les intersections. Ces objectifs sont réalisés à l'aide de l'introduction d'un nouveau type de robots, les Thymios signaux. Pour les intégrer, j'ai développé en réponse des limitations découvertes dans la première étape deux principaux outils nécessaires, la carte interne de chaque Thymio voiture et un protocole de communication entre Thymio signaux et Thymios voiture.

La troisième étape se concentre majoritairement sur l'utilisation des outils développés, pour implanter le contrôle des carrefours développé par M. Dubois, et permettre la mesure du système pour le contrôler d'une façon coordonnée. Dans cette partie, j'ai, à l'aide des théories du trafic routier normal, essayé de mettre en place des techniques d'optimisations actuelles.

Pour tester ces différents points, Nous avons créé une carte plus grande permettant d'utiliser tous les points développés précédemment sur le contrôle du trafic et des carrefours.

2 Description du projet

2.1 Le robot

Le Thymio est un petit robot éducatif développé à l'epfl. Il est conçu pour être abordable, robuste et simple à utiliser. Pour se déplacer, il utilise 2 roues indépendantes et plusieurs capteurs infrarouges qui lui permettent d'observer le sol et les obstacles. Plus précisément, 2 capteurs sont situés dessous et sont dévolus au suivi de ligne. Il possède un micro (seulement pour détecter l'intensité du son) et un haut-parleur. Il a aussi 5 boutons capacitifs dessus et de nombreuses leds RGB et mono-couleur. C'est un robot en développement continu et il peut depuis récemment utiliser ses capteurs de proximité pour transmettre des informations aux Thymios environnements.

2.2 Environnement de programmation

Deux possibilités existent pour programmer le robot. Une version vpl (Visual Programming Language) permet de programmer le robot de façon visuelle mais ne permet pas de prévoir de comportement complexe à cause des restrictions créées pour faciliter sa simplicité d'accès. L'autre possibilité consiste à utiliser Aseba Studio, un logiciel propriétaire, qui permet de programmer le microcontrôleur avec un langage simplifié sans les contraintes de programmation de base d'un robot. Par contre ce langage possède aussi plusieurs restrictions tel que l'utilisation obligatoire des types entiers 16[bit] et ne permet pas la définition de types avancés comme les structures. Ce langage utilise la logique événementielle, c'est à dire que le bout de code ne sera exécuté que lorsqu'un événement spécifique est lancé. Pour ne pas poser de problèmes avec le déroulement temporel, il nous faut limiter les calculs prenant trop de temps. [5].

2.3 Objectifs et division du projet

Comme il a été mentionné dans l'introduction, le but de ce projet est de créer et assembler toutes les infrastructures nécessaires à un réseau de transport pour un quarantaines de Thymios. Il doit s'inspirer du réseau routier réel, et posséder des routes, des intersections et des parkings. Il doit permettre aux Thymio de se déplacer simultanément d'un point à un autre sans risquer de collision ou de problèmes tout en respectant le code de la route de notre réseau.

Vu la taille de la tâche à accomplir, le projet a été divisé en deux. La partie locale est réalisée par Loïc Dubois [1] et la deuxième par moi.

- **Partie locale:** Se concentre sur le mouvement du robot
 - La poursuite de ligne et la lecture des codes-bar
 - Evitement d'obstacles et de collisions
 - Design des intersections (en collaboration)
 - Contrôle des intersections (au niveau des trajectoires)
- **Partie globale:** Focus sur les flux de robots
 - Flux de trafic fluide dans la carte
 - Design des intersection (en collaboration)
 - Contrôle des intersection (au niveau des flux)
 - Décisions haut-niveau et coordination des intersections

3 Réseau minimal fonctionnel

Dans cette section, je décris les besoins et les choix qui ont mené au design des différentes fonctions de notre réseau. Ce travail a été réalisé avec M.Dubois car cette définition influence le projet de manière globale. Le but étant qu'après avoir défini les caractéristiques et fonctions du réseau, nous puissions avoir un réseau simple et fonctionnel qui nous permette d'avancer individuellement les différentes étapes de nos projets.

3.1 Fonctions nécessaires

Pour faire fonctionner un réseau de transport, nous devons réaliser et assembler différentes composantes qui sont toutes indispensables à son fonctionnement. Le réseau doit contenir des routes permettant aux Thymio de se déplacer d'un point à un autre d'une façon fluide. Il nous faut aussi définir des intersections pour avoir un réseau contenant plus de 2 emplacements. De plus, Les Thymio doivent être capable de recevoir des informations sur les lieux dans lesquels ils se trouvent et sur les type de route dans lesquels ils se déplacent. Il faut aussi les programmer pour que le réseau puisse être utilisé simultanément par plusieurs robots voitures. Les collisions doivent être anticipée et évitées

Toutes ces fonctionnalités doivent être faites d'une façon simple et légère pour garder de la place dans la logique des robots et pour les complexifications futures. Elles font tout autant parties intégrante de la partie globale du projet car elles influent sur le système dans sa totalité.

Résumé des fonctions nécessaires à un réseau de transport de Thymio

- Routes
- Intersections
- Parking
- Technique d'information sur les lieux et les routes
- Comportement des voitures Thymios
- Gestion des collisions

3.2 Routes

Pour naviguer dans le réseau, les robots doivent pouvoir suivre des routes droites et des virages. Elles doivent connecter les lieux de la carte d'une façon bidirectionnelle et permettre aux Thymio de s'y déplacer d'une façon fluide.

Pour s'orienter, les Thymio possèdent deux capteurs infrarouges qui leur permettent de détecter la lumière réfléchiée par le sol et d'autres capteurs dirigés horizontalement pour la détection d'obstacles.

Avec M.Dubois, nous avons identifié ces trois possibilités.

Une ligne classique étroite et noir qui est la technique classique de suivi de ligne. Elle pose comme problème que le suivi n'est pas très précis car il ne change de direction que lorsqu'un de ses capteurs indique une sortie de ligne.

Un espace blanc entouré de noir matérialisant les limites de la route. Cette technique pose les

mêmes problèmes que la précédente. Une variante possible serait d'utiliser des murs physiques et ainsi utiliser les capteurs horizontaux mais cela demanderait un plus grand travail de fabrication.

En utilisant les résultats du projet Thymio light painting [4], c'est-à-dire, en utilisant une ligne composée de dégradés de gris et calculer la direction à prendre en fonction du niveau de gris mesuré. Cette technique n'utilise qu'un des deux capteurs de sol pour suivre précisément la trajectoire.

Toutes ces possibilités sont résumées dans le tableau 1

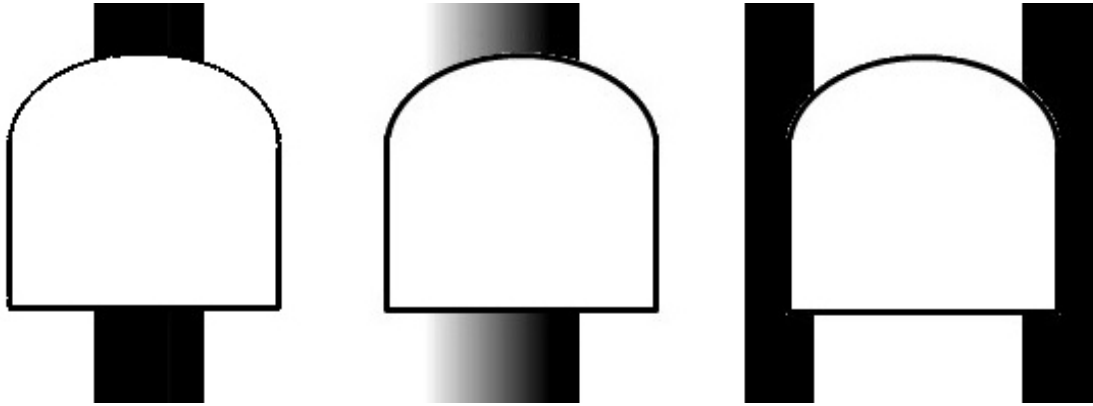


Figure 1: Illustration des différentes possibilités de routes

Table 1: Possibilités

Méthode	Pour	Contre
ligne étroite et noir	Suivi de ligne par défaut	Utilise les 2 capteurs de sol
Murs noirs	-	Même chose que la précédente
Dégradé de gris	Précis et n'utilise qu'un seul capteur	-

Nous avons choisi de les représenter avec la troisième possibilité, c'est à dire le dégradé de gris car elle permet d'utiliser le second capteur pour la détection d'informations (voir point 3.5) et qu'elle est la plus précise.

3.3 Les intersections

Pour réaliser un réseau contenant plus de 2 emplacements, nous devons choisir et réaliser au moins un type d'intersections.

Elles doivent permettre une sélection de la direction précise et éviter les situations de bouchons. Nous avons identifié trois possibilités.

Un carrefour blanc dans lequel les comportements des robots sont préprogrammés en fonction des choix de directions, cette technique impose une grosse gestion du flux pour éviter les bouchons et les collisions ainsi qu'une grande précision pour éviter les pertes de trajectoires.

Un carrefour type aiguillage de train avec une ligne continue pour suivre la trajectoire compliquée et un comportement préprogrammé pour la trajectoire droite (Voir figure 2). Par contre cette technique ne permet pas des carrefours en croix, mais seulement en T.

Un rond-point utilisant les aiguillages pour gérer les sorties et les entrées, qui permet un choix de

n'importe quel direction, même le demi-tour et permet de prévoir les instant critiques de collisions ou de bouchons mais qui nécessite une grande surface.

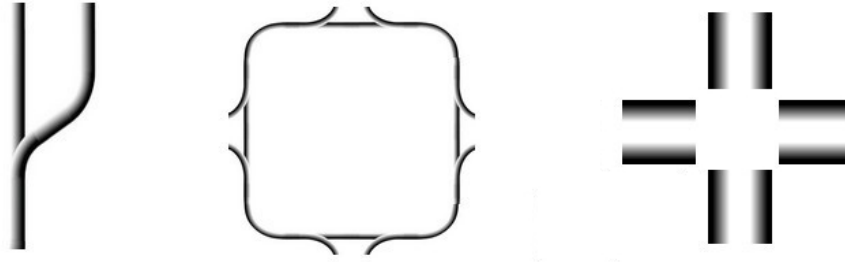


Figure 2: Illustration des types d'intersections possibles

Table 2: Résumé des points positifs et négatifs des intersections possible

Intersection	Pour	Contre
Aiguillage	Simple	Limité à 2 sorties
Rond-point	Generalisation de l'aiguillage	Grande taille
Carrefour	Petit	Besoin de contrôle et de précision

Nous avons donc choisi d'intégrer les ronds-points car ce type d'intersection est le plus simple et le plus efficace en termes de flux et de collisions. Ses inconvénient principale est sa taille.

3.4 Parking

Pour réaliser un réseau de transport, il faut aussi choisir la forme de ses parkings.

J'ai imaginé deux types de parking.

Le premier un parking en épis, qui utilise les aiguillages pour sortir de la route mais nécessite aussi de revenir en arrière pour repartir.

Le deuxième un voie de gare avec deux aiguillages pour sortir et retourner sur la route. Son inconvénient principale est qu'il est de type FIFO (First-in, First-out, premier entré, premier sorti)

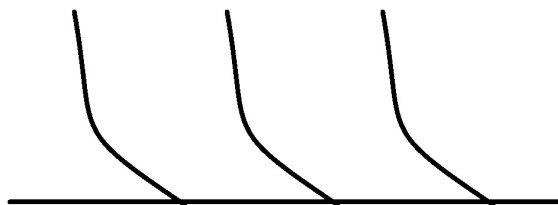


Figure 3: Place en épis

Pour des raisons d'économies de ressources, j'ai choisi d'utilisé le type voie de gare avec un éventuel développement futur des places en épis.



Figure 4: Voie de gare

3.5 Technique d'information sur les lieux et les routes

Pour comprendre la situation dans laquelle il se présente, le Thymio doit pouvoir recevoir des informations sur son emplacement immédiat. Par exemple, pour comprendre qu'il se trouve à l'entrée d'un rond-point ou sur une route rapide.

Ces informations doivent être localisée, car s'il ne la reçoit pas au bon moment, il ne pourra pas réagir correctement. Plusieurs technique sont possible, toute utilisent les capteurs infrarouges, en mode communication ou normal.

Premièrement, inspiré du projet de light painting, le code-barres sur la trajectoire du deuxième capteur de sol (voir partie 3.2), cette technique est simple à mettre en place et efficace

Eventuellement utiliser les capteurs horizontaux, mais cela imposerait de mettre les code sur les murs, donc plus difficile à fabriquer et de rend la lecture plus sensible à la lumière.

Nous pourrions aussi utiliser la communication inter-Thymio, elle est multidirectionnelle donc il serait difficile d'être sûr de la localisation des messages.

La seul solution praticable est la technique utilisée dans le projet de light-painting.



Figure 5: Barcode scanning

Après les tests réalisés par M. Dubois [1], le design du code-barres s'est fixé sur un bit de synchronisation et 3 bit d'informations car au-delà la fiabilité devient mauvaise. Ce qui nous laisse $2^3 = 8$ codes disponibles. Cette restriction impose l'utilisation de ces codes comme des informations très générales, juste des catégories de routes et des limites de vitesses (voir tableau 5). Des méthodes plus complexes seront nécessaires pour pallier à ces limitations (voir la partie 4.2 sur la carte interne).

3.6 Comportement des Thymio

Nous avons décidé d'implanter un comportement basé sur le hasard. A chaque fois qu'un robot est confronté à un choix, il décide au hasard lequel suivre. Ce comportement nous permet de rencontrer

la plupart des situations possible. Il est aussi possible de forcer les décisions des Thymio afin de rencontrer systématiquement la même situation.

L'intégration de la carte interne (Point 4.2) laisse la possibilité d'intégrer un algorithme de path-finding mais les capacités des robots ne sont pas suffisantes pour l'ajouter à toutes les fonctionnalités.

3.7 Gestion des collisions

Pour avoir un réseau avec de multiples Thymio, il faut avoir une gestion correcte des collisions.

M. Dubois les évite à l'aides des capteurs horizontaux infrarouges. Il est nécessaire de faire attention au design des différentes intersections pour éviter au maximum les situations difficiles. Une autre solution est d'ajouter un système de contrôle qui gère les priorités à l'avance et donc évite toutes les situations difficiles.

3.8 Prototype, tests et analyse

Comme première carte de tests, nous avons choisi un design simple permettant de montrer la viabilité de nos choix. Elle contient deux parking et un rond-point central. Le test était constitué d'une quinzaine de Thymio circulant au hasard tous lancé depuis un des parking. Leurs LEDs ont été réglées pour indiquer le type de zone dans lesquels ils se trouvent.

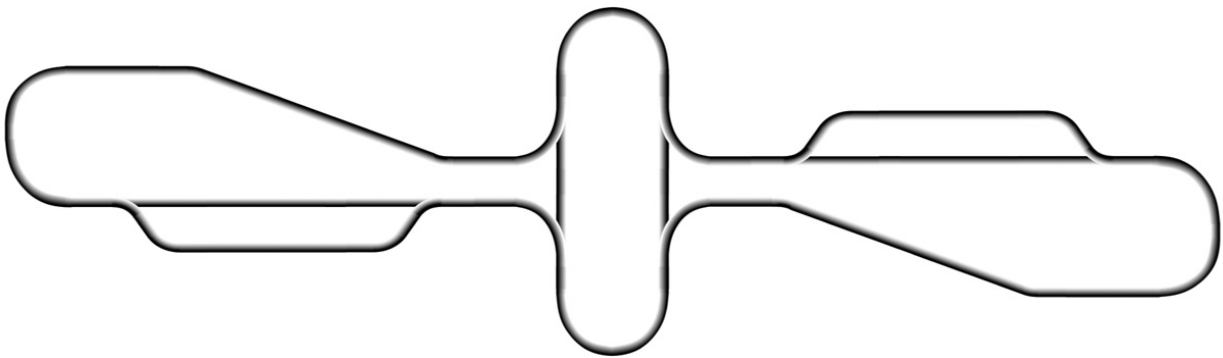


Figure 6: Carte prototype

Observations :

Une grande densité de robots au rond-point central. Elle est due au comportement au hasard qui laisse 50 % de chance de sortir donc beaucoup de chance d'y rester plusieurs tours et que c'est le seul point accessible depuis les parking

Dans certains cas, on voit apparaitre des collisions entre Thymio aux sorties de parking. Ça vient du fait que les deux voies ne sont pas assez éloignées donc que les capteurs des robots ne peuvent pas se détecter avant la collision. Deux solutions sont possibles, modifier le design des parkings ou ajouter un Thymio signal qui contrôlerais les sorties.

3.9 Conclusion de la section 3

Pour créer un réseau fonctionnel, j'ai en collaboration avec M. Dubois défini les différents points nécessaires à assurer un flux fluide de Thymio dans un réseau de transport. Le design définit ressemble fortement aux réseaux réels de chemins de fer et utilise des technique développées dans le projet de Light painting [4].

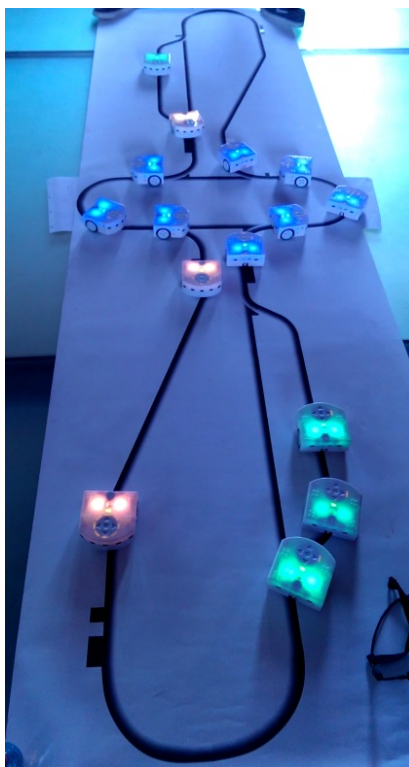


Figure 7: Test de la carte prototype avec une quinzaine Thymio. En Bleu ils sont dans le rond-point, en vert dans le parking et en rouge sur la voie-rapide

Résumé du design

- Ligne à dégradé de gris à sens-unique
- Intersections de type aiguillage
- Rond-point
- Parking comme une voie de gare
- Comportement au hasard
- Collisions gérées par les capteurs de proximités

4 Les complexifications et améliorations

Pour cette partie je décris les principales améliorations codées pour l'amélioration des Thymio-voitures. Il s'agit de la carte interne et de la communication inter-Thymio. Ces deux complexifications se sont imposées car en plus d'être un objectif additionnel pour la carte interne, elles permettent de réaliser les objectifs malgré les limites rencontrées lors de la section précédentes.

Ces complexifications devront de plus fonctionner avec le carrefour implémentés pas M. Dubois (voir [1]).

4.1 Gérer les intersections et la coordination

Maintenant que le réseau fonctionne, il faut pouvoir le mesurer et le contrôler. Pour réaliser ces objectifs, il faut utiliser un moyen de communication avec les Thymio voitures.

La bibliothèque de base du Thymio comprend une possibilité de communication en utilisant les capteurs IR horizontaux pour transmettre des nombres aux robots à portées. L'utilisation de ce type de communication est problématique car tous les robots à portée le reçoivent et n'ont pas de moyen d'en connaître l'origine. Cette communication s'apparente à un broadcast local.

Le Thymio wireless récemment développé est une alternative, et possède une portée suffisante pour tout le circuit, mais la jeunesse de ce système ne permet pas d'en disposer d'une quarantaine et n'est donc pas praticable pour faire un test à grande échelle comme prévu.

Donc la communication se fera à l'aide de l'infrarouge

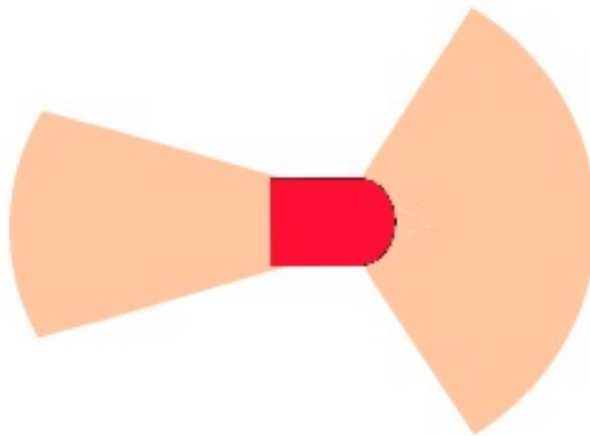


Figure 8: Zone de reception et de transmission d'un thymio

4.2 Raisons du développement de la carte interne

Le fait que la communication IR corresponde à du broadcast (avec quelques limitation) pose problème, lorsqu'un Thymio voiture reçoit un ordre (par ex: Stop!) il doit pouvoir interpréter si cette ordre lui est bien destiné.

Comme première solution, j'ai pensé utiliser les codes-barres pour coder les lieux dans lesquels ils se trouvent et donc simplement ajouter aux ordres le lieu de destination. Pour des raisons de fiabilités, les codes sont limités à 3 bits ce qui laisse juste assez de possibilités pour y mettre le type d'élément

devant lequel il se trouve.

Une autre possibilité est d'avoir une caméra qui filme en permanence le réseau et interprète à l'aide de symbole (par ex. des QR-codes) la position et le nom des Thymio à tout moment. Cette solution impose beaucoup de matériel et une gestion centralisée du système. Elle est réalisable en rajoutant cette fois le nom ou le numéro adressé lors d'une communication.

La dernière imaginée est d'implémenter une représentation interne de la carte à tous les Thymio voitures qui leur permettrai de connaître leur position à tout moment et serait mise à jour à chaque lecture de codes barre.

Le choix s'est porté sur une carte interne car elle permet un fonctionnement décentralisé du système.

4.3 Fonctionnalités et réalisation de la carte interne

La carte interne doit être implémentée dans chaque Thymio voitures.

Elle doit donc prendre peu de ressources et peu de temps de calculs car elle fonctionne en même temps que les fonctions de bases du Thymio.

La carte interne doit permettre à chaque robots de connaître sa route actuelle.

Elle doit être capable de détecter les erreurs de lectures

Elle doit pouvoir se réinitialiser lors de situation prévues

Représentation On veut représenter une carte d'une dizaine de nœud en sachant qu'un Thymio n'a pas plus de 500 variables entières à disposition.

La méthode la plus économe en mémoire et ressources consiste à représenter une carte par un ensemble de nœud connecté entre eux par des chemins. Cette représentation correspond à un graph orienté.

Le seul type utilisable dans Aseba Studio, est l'entier 16 bit. Étant donné que la taille des cartes ne dépassera pas une vingtaine de nœuds, J'ai choisi d'utiliser un tableau d'entier pour représenter la carte. Je ne représente que les connections sortantes dans le graph, car cela divise par 2 le nombre de variable nécessaires et que la représentions de l'ordre des entrées sorties nécessiterai elle aussi plus d'une variables par noeuds. Cette limitation impose un traitement particulier de certains cas dans une carte plus complexe que la carte prototype.

La carte est représentée avec les variables suivantes,

La variable map contient le graph lui-même

la variable maptype contient le type de nœud

la variable orientedIntersection contient l'ordre des sorties des intersections. Elle n'est définie que pour les intersections, donc toutes les intersections doivent se trouver au départ du graph.

Définition des variable de la carte interne pour la carte prototype

```
#bit of map[x] = 1 if connected with the corresponding node.  
map[0]=0b110  
map[1]=0b001  
map[2]=0b001
```

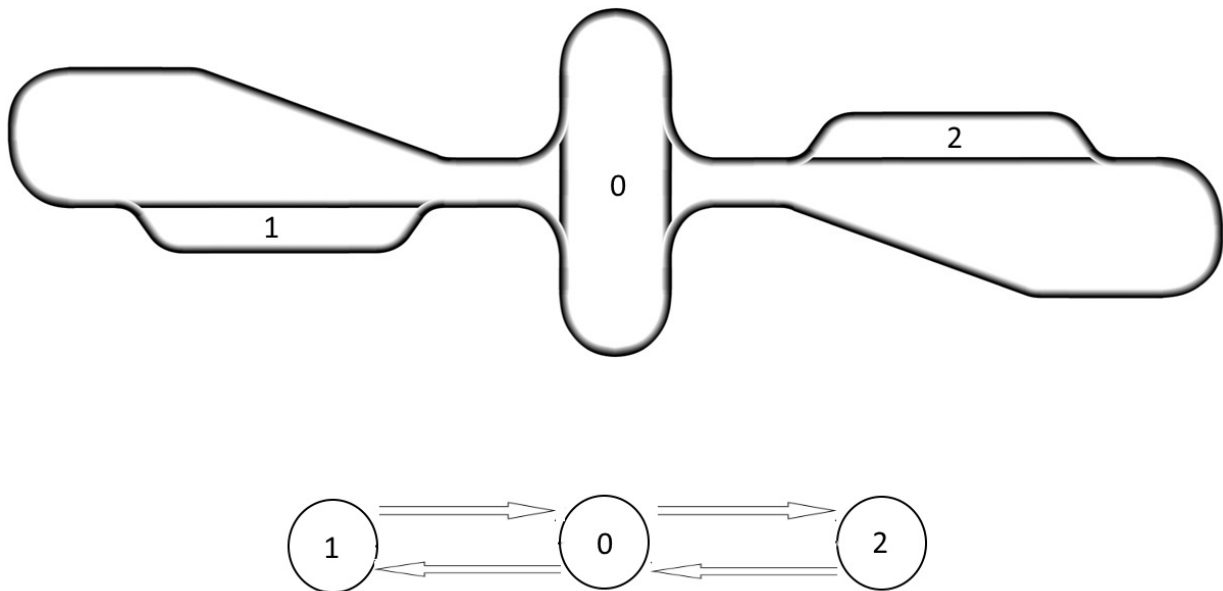


Figure 9: Graph orientés de la carte prototype, dans ce cas simple le graph positif correspond au graph négatif

#type de noeuds:

#00=location

#01=round-about

maptype[0]=0b01

maptype[1]=0b00

maptype[2]=0b00

#Information on the order of the nodes in an intersection 001 010 011 100

#The bit coding represents the nodes connected with 3 bits

(8 possibilities of connections)

but it is saved on 16 bit therefore maximum 5 nodes per intersections with

001 # == smallest connected node and 000 indicates null

orientedIntersection[0]=0b 010 001

Algorithme de recherche du noeud correct

1. Parcours du tableau map[noeud actuel]

Stockage dans nodTab de tous les noeuds connectés

Stockage de l'indice du noeud précédent

```
for i in 1:N_noeud
    si map modulo 0b10 == 1
        nodTab[j]=i
        si oldNode == i
            oldIndice == j
        end
```



```

        end
      map << 1
    end

```

2. Recherche dans `orientedIntersection[noeud actuel]` de la branche correspondante à la branche d'entrée dans l'intersection

```

while(test)
  si orientedIntersection modulo 0b1000 == oldIndice
    test=false
  end
  orientedIntersection << 3
end

```

3. Parcourir `orientedIntersection` jusqu'à concurrence du nombre de sorties évitées.

```

for i in 1:Nb_sorties_évitées
  orientedIntersection << 3
end
newNode = nodTab[orientedIntersection]

```

La limite de cet algorithme, c'est qu'il ne permet que d'avoir des doublets d'entrées-sorties, qui sont connectées au même nœud. Cela oblige d'avoir comme seul chemin complexe des intersections connectées entre elles.

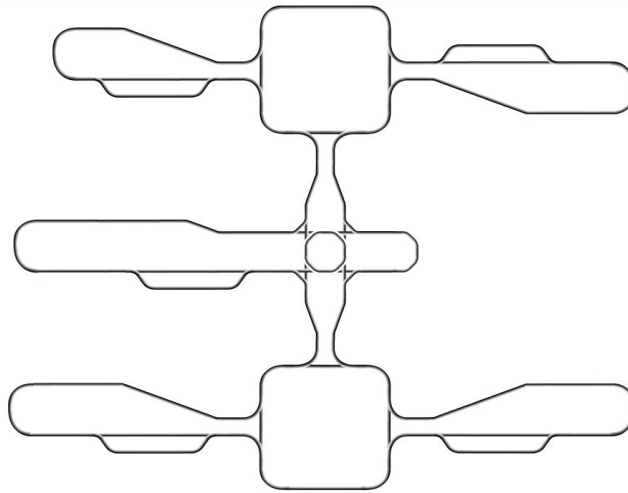


Figure 10: Carte final avec la carte interne de base

Pour rendre cet algorithme général, il faudrait aussi implémenter les différentes entrées des intersections dans un autre tableau d'entier.

Ce tableau supplémentaire utiliserai plus de mémoire et rajoute du traitement car la variable `orientedIntersection` ne peut pas gérer plus de 5 entrées-sorties.

Vu que les situations limites sont prévisible en fonctions du design de la carte, au lieu de modifier en profondeur l'algorithme, on peut simplement rajouter du traitement pour gérer correctement ces cas précis.

Après avoir testé les deux possibilités, j'ai implémenté la deuxième car avec toutes les complexifications apportées, la taille maximum du bytecode est presque atteinte et que la deuxième consomme légèrement moins de ressources.

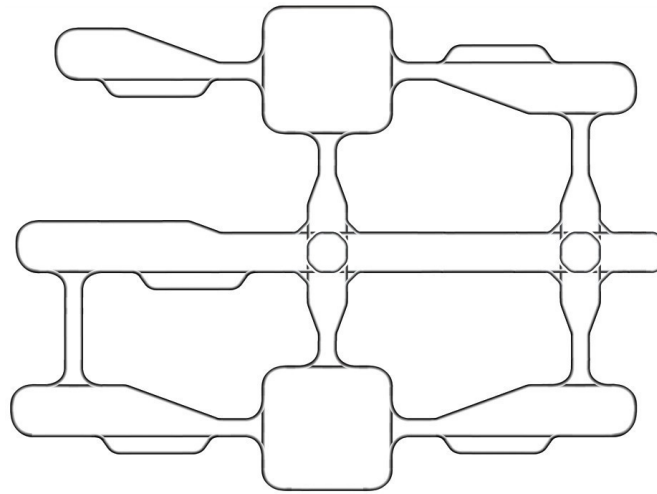


Figure 11: Carte final avec la carte interne améliorée

Cas particuliers Deux cas sont rendus difficiles par la non implémentation du graph orienté négatif.

Le cas d'une boucle contenant un branchement.

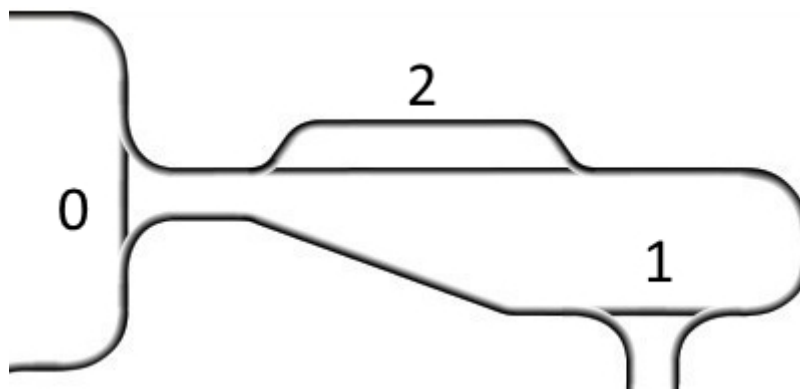


Figure 12: Graph non-orientés de la carte prototype

Ici le problème vient du fait que la sortie ne correspond pas au nœud d'entrée, ce qui ne permet pas à l'algorithme de trouver l'indice d'entrée du tableau.

ce problème a été résolu en rajoutant une condition qui permet d'aller chercher le nœud d'entrée en le recherchant parmi tous les nœuds de sortie.

Le cas d'une sortie d'un rond-point pour aller dans un parking en passant par un branchement (Voir figure 13).

Ce cas est plus difficile. Il vient du fait que lorsque l'on arrive depuis le cross road, le nœud numéro 1 n'est même pas mentionné puisque le robot n'a pas de possibilité de changer de direction. Ce cas rend la recherche du point encore plus compliqué puisqu'il faut la rechercher 2 crans plus loin.

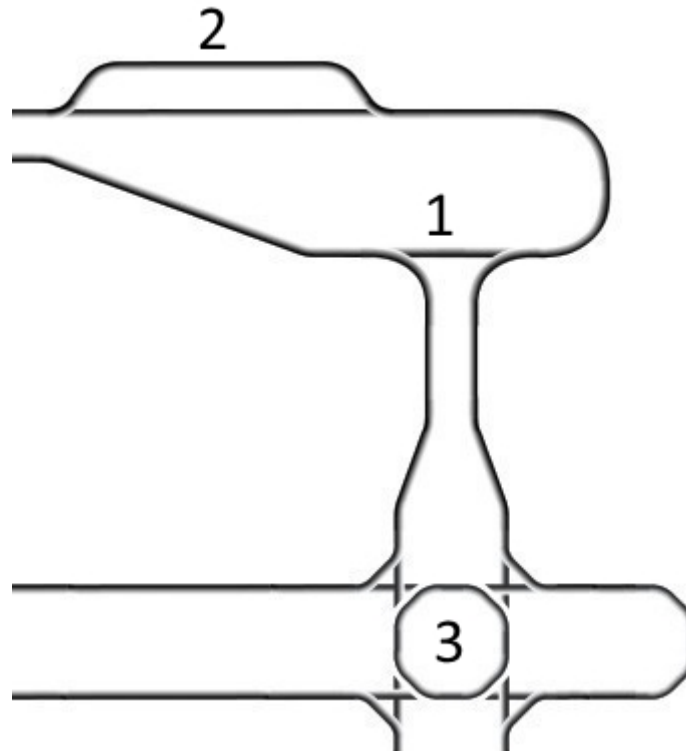


Figure 13: Graph non-orientés de la carte prototype

Pour résoudre ce problème, j'ai légèrement triché sur la définition de la carte. En effet, je mentionne dans la carte le nœud numéro 1 comme une des sorties du carrefour alors que ce n'est pas le cas (Il n'y a pas de code-barres dans cette situation) et je traite cette erreur de position en relançant le programme de recherche qui la trouvera la bonne solution.

Le Thymio est capable de connaître à tout moment sa position, si sa position de départ est connue et qu'il ne fait pas d'erreurs de lecture. Pour résoudre des éventuelle erreurs, nous avons rajouté un second code-barre dans les parking avec lequel les thymio peuvent remettre à zéro leur carte.

4.4 Communication

La communication entre les Thymio utilise les capteurs et émetteurs infrarouge horizontaux. Elle fonctionne entre les pulse émit (toutes les 100[ms]) pour mesurer la lumière environnante, et permet de transmettre un entier de 11 bit par cycle. Sa portée va jusqu'à 40 cm et est dépendante du niveau de luminosité ambiante. Ses capteurs ne sont pas placé à 360 degrés, mais disposent d'une zone de transmission sur l'arrière et d'une autre sur l'avant (voir ??).

Lors de la réception d'une transmission, un évènement est automatiquement émit, il est donc aisé de programmer sa réaction dans le robot.

Dans le design initial, les intersections étaient uniquement gérées par les capteurs de proximités, les tests ont montré que lors de certaines situations, cela pouvait mener à des bouchons et des collisions

Pour résoudre ces problèmes, j'ai imaginé 2 possibilités:

La coordination inter-thymio: Le fait que les Thymio communiquent entre eux en s'échangeant

leur position et décident en fonction des règles et principes programmés, leur priorité comme dans un carrefour réel. Des erreurs risquent d'apparaître avec cette solution, a commencé par la distance maximal de transmission un peu faible pour aller d'un côté à l'autre des carrefours désigné par M.Dubois. Elle demande de plus beaucoup de ressources du microcontrôleur.

Thymio spécialisé en signaux: Utiliser des Thymio exclusivement comme des feux ou des policiers, c'est à dire qu'ils décident qui passe quand. Cette solution permet de mettre un robot signal au centre du carrefour ce qui résout le problème de la distance de transmission et leur laisse le soin de calculer les meilleures possibilités de parcours ce qui économise les ressources des Thymio-voitures.

J'ai donc implémenté les robots-signaux

4.5 Les protocoles de communication inter-thymio

Le but premier des robots signaux est de transmettre l'état de toutes les routes qu'il contrôle le plus vite possible. Après, pour adapter ses décisions à l'évolution de la situation, il doit être capable de mesurer l'état de ses routes.

Deux cas de communication coexistent

- Communication unidirectionnelle du signal à toutes les voitures à portée (CS_TELL dans le thymio signal et CS_EAR dans le thymio voiture)
- Communication bidirectionnelle entre le signal et un des Thymios (CS_EAR et CS_ASK dans le thymio signal et CS_TELL dans le thymio voiture)

Pour différencier ces deux types de communications, j'ai implémenté une machine d'état dans chacun d'eux:

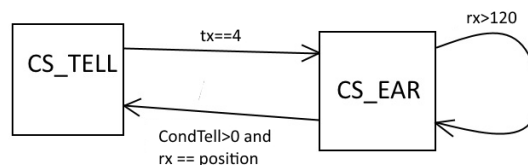


Figure 14: Machine d'état du Thymio voiture

Communication unidirectionnelle J'ai donc implémenté ce protocole de disque de broadcast qui donne à tour de rôle directement l'état des routes (Ouvert ou fermé) Un disque de broadcast consiste en un tableau d'informations fixe (Dans notre cas il est renouvelé toutes les 30 secondes environ en fonction du cycle choisi pour le contrôle) et qui envoie ses informations une par une à intervalles régulières en le parcourant dans l'ordre. Le principe est que lorsqu'un utilisateur cherche une information, il écoute et sélectionne la bonne. Il s'agit de la méthode la plus rapide pour envoyer des informations différentes à de multiples destinataires. Mais qui n'est pas très économe en énergie.

Pour cela j'ai défini ces conventions:

- La precedente position + 20 = vert
- La precedente position + 40 = rouge

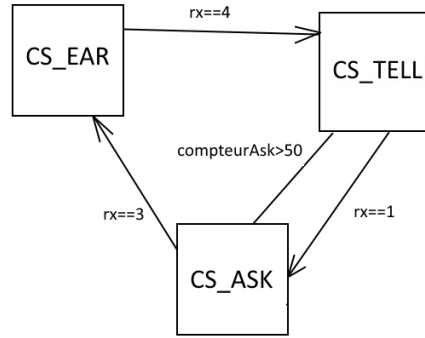


Figure 15: Machine d'état du Thymio signal

Lorsque le Thymio détecte son ancienne position + 40, il stocke son ancien état et s'arrête. S'il reçoit sa position + 20, il redémarre et reprend son ancien état.

Les Thymio peuvent donc comprendre directement ces informations et réagir fonction très rapidement.

Communication bidirectionnelle Le but de cette communication est que lorsque qu'un Thymio se manifeste, il puisse envoyer des informations au signal pour qu'il puisse réagir en fonction ou simplement d'en faire des statistiques.

La difficulté de la transmission de plusieurs informations à la suite consiste à la synchronisation et au temps de réaction des deux robots. Si l'information change trop vite, le signal risque de ne pas avoir le temps de les stocker.

La solution implémentée consiste en un début de communication qui change les états des deux Thymio concernés. Un fois lancée, à chaque fois que le Thymio voitures envoie une information, le signal répond un message de contrôle différent. Ce qui permet au Thymio voiture de passer à l'information suivante. Une fois toutes les informations envoyées, les Thymio repassent dans leurs états normaux et le signal peut traiter ses données.

Table 3: Communication bidirectionnelle

Time	Signal	Voiture (En parenthèse le code utilisé)
1	-	Hello (1)
2	Hello (1)	-
3	-	Début de communication (3)
4	Compris1 (11)	-
5	-	Position + 60
6	Compris2 (10)	-
7	-	Old position + 60
8	Compris1 (11)	-
9	-	Fin de communication (4)

Ce type de communication fonctionnera directement si l'on veut rajouter des informations liées au comportement du Thymio par exemple. Par contre elle permet que de mesurer le Thymio qui passe à ce moment devant le signal. Elle ne permet pas de mesurer la quantité de Thymio sur une route complète.

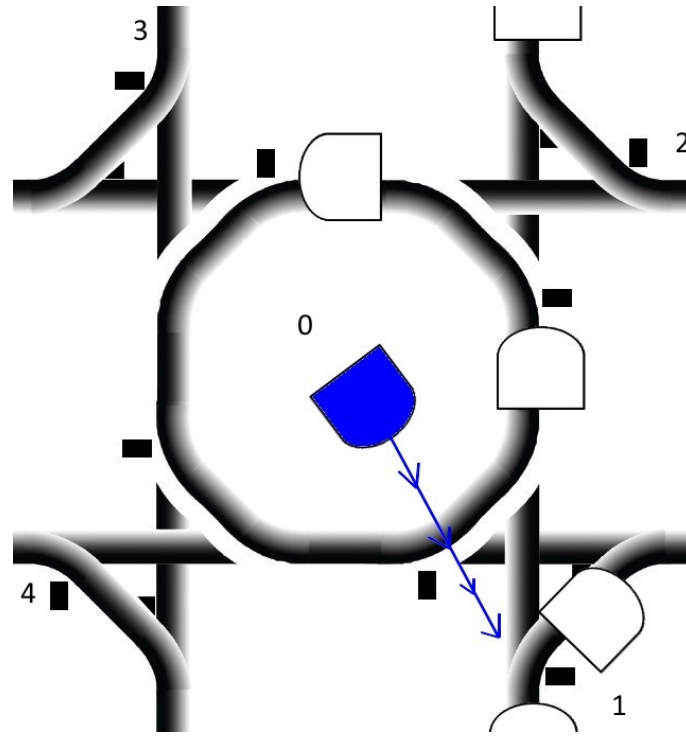


Figure 16: Exemple de carrefour contrôlé, le noeud du carrefour est le 0 et les entrées sont respectivement les 1,2,3,4

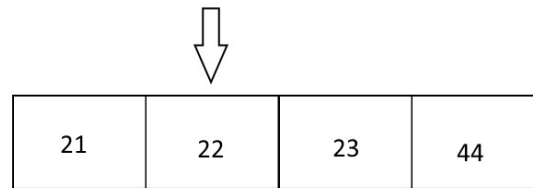


Figure 17: Exemple du disque de broadcast pour le carrefour de la figure 16. la flèche change de position toutes les 200[ms] et parcourt le disque en 800[ms]. Les informations dans le disque ne changent que lorsque la logique du signal décide qu'il est temps d'ouvrir une autre route

Mesures du nombre de Thymios sur une route

Pour faire fonctionner un contrôle adaptatif des carrefours, il faut pouvoir mesurer la quantité de Thymios sur la même route. Dans le cas d'une file de Thymio voitures attendant au feu, puisque la zone de transmission de capteurs est en grande partie bloquée, il est difficile au signal de les questionner directement.

La solution implémentée est un ping à incrément. À la fin d'une communication bidirectionnelle, le Thymio voiture émet automatiquement sa position + 100. Tous les Thymio voitures sur la même position vont soit réémettre le même nombre (s'ils ont déjà répondu à ce type de requête récemment) soit l'incrémenté de 20 et l'émettre de suite. Comme cela, le premier Thymio voiture finira par émettre le nombre de Thymio derrière lui.

La valeur de base est 100 + position et l'incrément de 20 comme ça on évite d'avoir des valeurs similaires utilisées pour les autres informations communiquées et donc un risque de mauvaise compréhension.

Cette technique fonctionne et donne des résultats fiables, par contre on risque d'avoir des situations

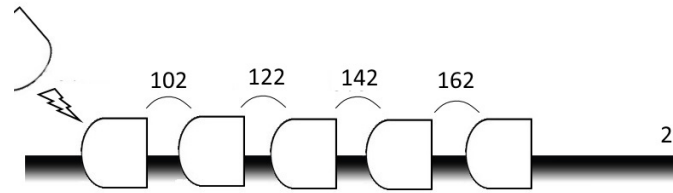


Figure 18: Propagation du ping, les voitures incrémentent le signal de 20 si elles sont sur la bonne route et qu'elles n'ont pas déjà incrémenté.

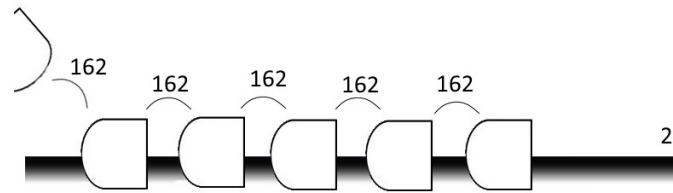


Figure 19: Retour du ping, les voitures retransmettent la plus haute valeur qu'elles reçoivent.

dans lequel les l'incrémentation se maintient en fonction de nouvelles arrivées sur la route.

Conclusion

La gestion des intersections a nécessité le développement de deux grandes parties, la carte interne et la communication inter-thymio. Elles ont été un challenge à cause des ressources limitées à disposition et ne sont viables que pour une taille de 16 nœuds au maximum pour la carte. Pour un éventuel travail futur, une possibilité serait de stocker une plus grande carte dans une mémoire externe et de ne charger que la partie que le robot utilise. Les protocoles de communication sont complets et utilisables sans difficultés à plus grande échelle, par contre la carte interne nécessitera des retouches en cas de situations différentes.

5 Contrôle et mesures du système

Dans cette section je vais décrire la base de la théorie du Traffic routier pour expliquer les différentes possibilités de contrôle du réseau modélisations et pour permettre la comparaison avec un réseau routier réel.

5.1 Théorie et modélisation de réseaux routier

La traffic routier se mesure et modélise de plusieurs façons, je vais ici décrire les mesures les plus simple représentant les phénomènes les plus caractéristiques d'un réseau routier. (voir [3] et [2])

Deux approches existent,

La macroscopique qui modélise le traffic comme un fluide. Il décrit le comportement du réseau à l'aide de variables continues calculé à partir de valeurs discrètes. C'est la méthode la plus simple.

la microscopique qui modélise le traffic jusqu'à l'échelle du conducteur et de sa voiture, par définition, il permet de reconstruire les résultats de l'approche macroscopique, mais demande beaucoup de ressources.

Variables

Le traffic routier se décrit à l'aide de ces trois variables.

- **La densité:** la quantité de véhicule par mètres.
- **Le débit:** Le nombre de voitures par secondes.
- **La vitesse moyenne:** qui est simplement le débit divisé par la densité.

5.2 Contrôle des parking

Pour résoudre le problème des collisions en sortie de parking, j'ai décidé de mettre des Thymio signaux pour gérer les priorités. Ils sont identique aux robots signaux normaux mais envoie juste la position + 20 ou + 40 selon la voie ouverte.

5.3 Contrôle des intersections

Au niveau du contrôle, les robots signaux sont utilisés comme des feux de signalisation. Il faut pour cela que les communications qu'ils dispensent soient correctement reçues sur toutes les routes d'arrivées dans le carrefour.

Nous avons essayé avec m. Dubois de trouvé un design de carrefour permettant de desservir avec un seule robot signal 4 entrées, mais à cause de la disposition de la zone de transmission et des rayons de suivi de ligne nous n'avons pas pu trouver de solutions.

Il aurait été possible de démonter les robots signaux pour en déporter les capteurs, mais le laboratoire m'a mis à disposition 4 Thymio wireless évitant ainsi ce moyen. Pour pallier à la zone discontinue, les robots signaux sont constitué de 2 robots posé l'un sur l'autre avec un configuré en esclave ne servant qu'à retransmettre les décisions.

Contrôle

Pour le contrôle des carrefours, je vais considérer une seule ligne ouverte par phase de cycle des feux. L'optimisation se fera grâce à la modification du cycle. Le calcul de plan de feu optimal nécessite quelques valeurs pratiques mesurées sur le circuit. Flux de saturation du carrefour :
C'est simplement le débit maximum de véhicules en cas de feu vert perpétuels.

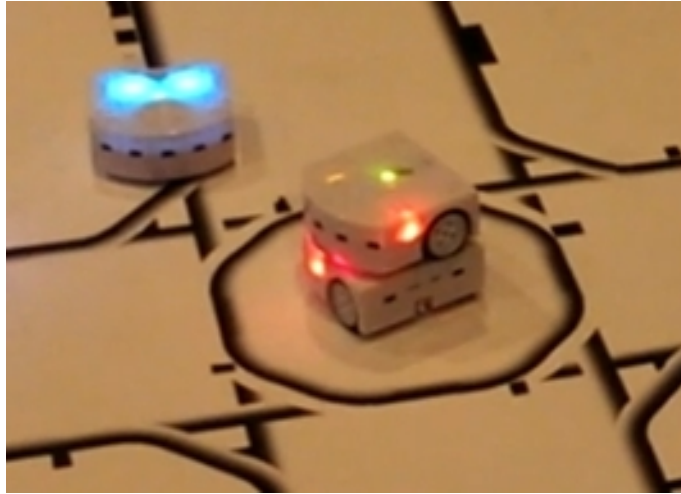


Figure 20: Placement des thymios signaux

Il est de 0,2 Thymio par secondes

Les temps d'établissement et d'extinction :

10 secondes d'établissement de flux.

15 secondes pour la libération du carrefour.

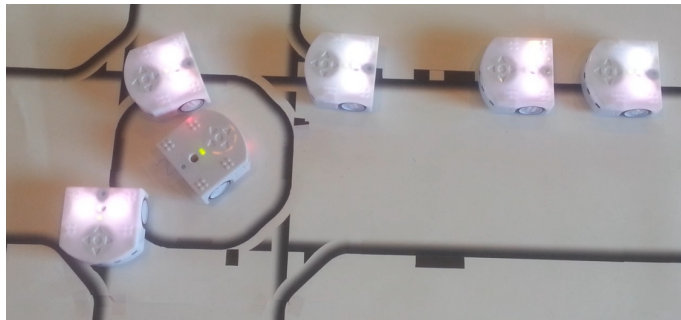


Figure 21: Mesures du débit de saturation

Plan de feux La méthode Webster [2] consiste à calculer le cycle optimum de feu. Elle utilise l'équation suivante pour cela.

$$C_o = \frac{1.5L + 5}{1 - \sum_{i=1}^{\phi} \frac{q_{ij}}{S_j}}$$

C_o = Le temps de cycle optimum [s]

L = Temps perdu total [s]

q_i = Débit sur la route i au feu vert

S_j = Flux de saturation

Le temps de cycle est le temps total du feu pour toutes les phases (Une phase est une route dans ce cas), il est donc distribué en fonction des besoins réels de débit par route. Le cycle est proportionnel

à la demande, en effet plus on change de phase souvent plus la part des pertes de temps deviens grande par cycle, donc pour satisfaire un grande demande, il vaut mieux réduire les pertes.

5.4 Programmation et implémentation

Types d'algorithmes

Plan de feux définis offline Selon la formule de Webster, je calcule plusieurs cycles optimaux avec des valeurs de demandes différentes et j'applique une distribution uniforme de ce cycle sur les phases, vu que le comportement des Thymio est basé sur le hasard.

Plan de feux adaptatif Basé sur l'article *Adaptive Traffic Lights Using Car to Car Communication* [6] Il s'agit aussi de calculer le cycle avec la formule de Webster, puis de la recalculer chaque cycle en fonction de la demande réelle.

5.5 Tests

Ce test a été réalisé pour tester toutes les fonctionnalités mise en place durant le semestre. Nous avons à notre disposition une quarantaine de Thymio, dont une bonne partie qui ne fonctionnait pas suffisamment bien pour être utilisées en tant que voiture. Les LEDs du dessus servent à indiquer le type de route ou le type de lieux dans lesquels ils se trouvent. Les Thymio ont été lancés depuis les différents parkings avec comme instruction de lire le premier code-barres pour initialiser la carte interne.

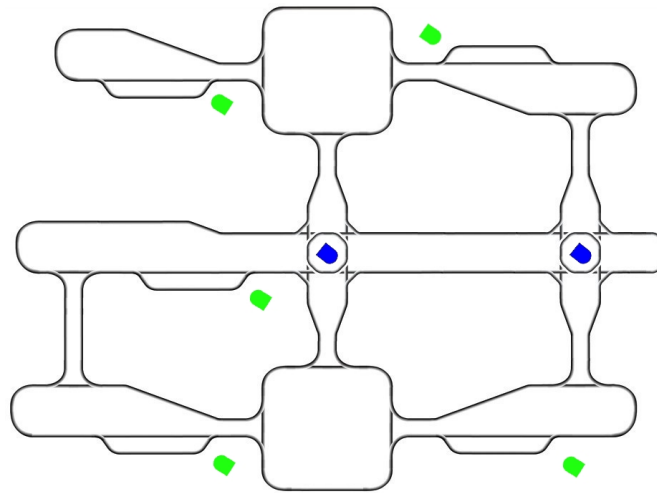


Figure 22: Configuration de la carte testée, en vert les Thymio parking, en bleu les Thymio signaux

Observations :

Nous avons pu voir et résoudre la plupart des problèmes dus aux signaux et les feux de parking résolvent comme prévu le problème des collisions au sorties.

Sauf le fait que leur portée diminue la nuit, surement à cause de l'éclairage. Après la tombée de la nuit, nous n'avons plus pu faire fonctionner correctement les signaux. Donc cela nous a permis d'observer que lors de l'absence de ces contrôles de nombreuses situations de bouchons apparaissent.

Ces problèmes de fiabilités nous ont empêchées de prendre les mesures prévues donc nous allons refaire des tests pour avoir ces résultats lors de la présentation du projet.

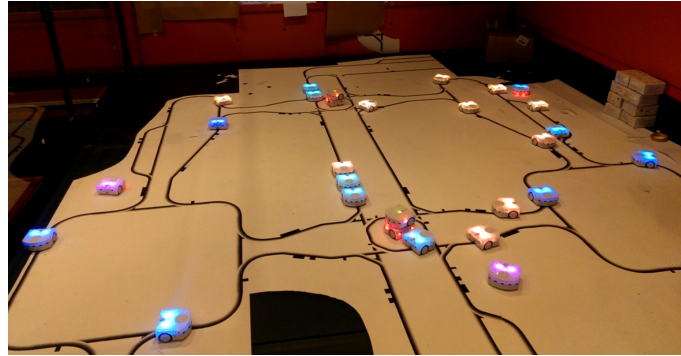


Figure 23: Test d'une trentaine de Thymio avec toutes les fonctionnalités implémentées

5.6 Conclusion de la partie contrôle

Dans cette partie, j'ai choisi et implémenté le contrôle des signaux du système, j'ai dû me contenté de n'implémenter que la solution standard aux problèmes de gestion des timings d'intersections par manque de temps au vu de la quantité du travail à fournir pour mettre ensemble toutes les parties (que ce soit au niveau de l'optimisation et simplement des bugs)

6 Possibilités futurs

- L'ajout d'un algorithme de path-finding, l'interet de cette amélioration serait de permettre au Thymio de trouver par eux-mêmes d'autres parcours pour aller jusqu'à leurs destinations et donc permettre de la coordination plus avancée. Mais les ressources computationnelle risquent de manquer fortement pour cette implémentation.
- La création automatique de statistique, qui utilisera les robots-signaux comme calculateurs de différentes informations utiles comme par exemple le temps d'attente moyen. Cela permettra de comparer les différentes techniques de contrôles et d'optimisations beaucoup plus facilement.
- La généralisation des Thymio wireless, ce qui améliorera énormément la fiabilité des thymio signaux et donc la fiabilité du réseau et rendra encore plus aisé la récupération de statistiques.
- La comparaison du comportement dynamique des Thymio avec le réseau routier réel, Comme la transmission des ondes d'arrêts dans une file de voiture.
- L'implémentation et les tests d'autres types de contrôles (centralisé ou basé sur la coordination).
- La carte "sur demande", ce serait de stocker une plus grande carte dans une mémoire externe et de ne charger que la partie que le robot utilise. Ce qui permettrait l'utilisation d'un plus grand réseau sans nouveaux matériels.

7 Conclusion

Pour créer un réseau de transport adapté aux Thymio, nous avons implémenté toutes les fonctions nécessaires à sa réalisation. M. Dubois a travaillé plus la programmation de ce projet et moi je me suis concentré sur la partie globale. Pour assurer un flux fluide au réseau, nous avons sélectionné de la technique inspirée de la voie ferrée et des routes réels.

Les Thymio reçoivent des informations principalement par la lecture de codes-barres qui sont placés le long des routes. En raison du manque de complexité de ces codes, les Thymio utilise une carte interne pour interpréter leur position en fonction de leur parcours.

Je leurs ai aussi implémenté aussi des protocoles de communications pour intégrer le contrôle des intersections par des robots spécialisés, des robots signaux. Ces deux améliorations ont été difficiles à faire rentrer dans la logique des Thymio à cause de leurs faibles capacités mémoires. Elles ont nécessités des astuces pas très propres d'un point de vue informatique mais efficaces.

Les signaux ont aussi permis le management des timings en temps réels, ce qui permet d'en comparer les performances sur une carte de $20m^2$. Même si la fiabilité étant limitée retarde les conclusions de ces études.

Ce projet regorge de nombreuses possibilités d'améliorations, il nous a vraiment permis de développer nos approches et nos solutions. De nombreuses possibilités existent pour passer outre les limites imposées par l'architecture de ces robots et développer les différentes améliorations soulevées dans ce rapport.

Bibliography

- [1] Loïc Dubois. Thymio road network, local report. Technical report, Ecole polytechnique fédérale de Lausanne, LSRO, 2015.
- [2] Nicholas J. Garber. *Traffic and highway engineering 4th ed.* Stamford, CT : Cengage Learning, 2010.
- [3] Saïd Mammar. *Systèmes de transport intelligents*. Lavoisier : Hermes science, 2007.
- [4] Thymio Light Painting Project. <https://aseba.wikidot.com/en:barcodelightpainting>, April 2015.
- [5] Thymio. <https://www.thymio.org/>, April 2015.
- [6] Cristian Gorgorin Victor Gradinescu. Adaptive traffic lights using car to car communication. *IEEE*, page 6, 2007.

Appendices

A Tableaux de codes

Table 4: Liste des codes-barres

Code	Information
0	-
1	SPEED_LOW
2	SPEED_MED
3	SPEED_HIGH
4	A_BRANCHING
5	A_CROSSROAD
6	A_LOCATION
7	A_ROUNDABOUT

Table 5: Codes de communications

Code	Information
1	Bonjour
3	Début
4	Fin
10	ok1
11	ok2

B Code Thymio voiture

```

<!DOCTYPE aesl-source>
<network>

<!-- list of global events -->

<!-- list of constants -->
<constant value="500" name="GREY_REF"/>
<constant value="152" name="SPEED_LOW"/>
<constant value="264" name="SPEED_MED"/>
<constant value="416" name="SPEED_HIGH"/>
<constant value="400" name="GREY_THRESH"/>
<constant value="1000" name="OBS_PROX_THRESH"/>
<constant value="3000" name="OBS_PROX_THRESH_MAX"/>
<constant value="0" name="S_STOP"/>
<constant value="1" name="S_ROAD"/>
<constant value="2" name="S_READ"/>
<constant value="3" name="S_ACTION"/>
<constant value="4" name="S_EXIT"/>
<constant value="5" name="S_CONTINUE"/>
<constant value="6" name="S_BACK"/>
<constant value="0" name="A_BRANCHING"/>
<constant value="1" name="A_ROUNDABOUT"/>
<constant value="2" name="A_CROSSROAD"/>
<constant value="3" name="A_LOCATION"/>
<constant value="0" name="DS_NOTHING"/>
<constant value="0" name="DS_EXIT"/>
<constant value="1" name="DS_CONTINUE"/>
<constant value="2" name="DS_TURN_LEFT"/>
<constant value="4" name="DS_STOP"/>
<constant value="16" name="NB_NODE"/>
<constant value="0" name="CS_EAR"/>
<constant value="1" name="CS_TELL"/>

<!-- show keywords state -->
<keywords flag="true"/>

<!-- node thymio-II -->
<node nodeId="1" name="thymio-II">var state = S_STOP
var oldState = S_STOP
var decisionState = DS_NOTHING

var speedType = SPEED_LOW
var savedSpeed = SPEED_LOW
var savedSpeedInters = SPEED_LOW
var addSpeedLeft = 0
var addSpeedRight = 0

var controlFactor = 1
var intensityDiff = 0
var intensityObst = 0
var intensityDiffObst = 0
var white_threshold = GREY_REF+200

var counter = 0
var counterContinue = 0
var counterStop = 0
var counterRead = 0
var code = 0
var interval = 0
var readLocation = 0
var random = 0
var actionType = A_LOCATION

var timerON = 0
var incrementCounter=1

#####
#Variables for the intern map
var map[NB_NODE]
var maptype[NB_NODE]
var orientedIntersection[8]
var nodeTab[5]
var newNode = 0
var newNodeType = 4
var currentNode = 9
var oldNode = 9
var positionError = 0
var i = 0
var stick
var nodeCount=0
var exchange=0
var t=0
var indice
var oldIndice = -1
var counNode = 0
var test2=1
var countExit=1
var nbRond=0
var NbCont=2
#communication variables
var comm_state=CS_EAR
var last_ok =11
var l=0
var countPing = 1

#pos[0]= actual position

```

```

#pos[1]= old position
#variable for measurements
var count=1

#Map declaration
#bit[0 and 1] = node type + address in the table orientedIntersection
#bit[following] = 1 if connected with the corresponding node.
#
# 9876543210
map[0]=0b000000100010100
map[1]=0b001000001000100
map[2]=0b000000000101011
map[3]=0b000000010011100
map[4]=0b000001000001000
map[5]=0b000110000000000
map[6]=0b000110000000000
map[7]=0b000000000001010
map[8]=0b000000000000001
map[9]=0b000000000000001
map[10]=0b000000000000100
map[11]=0b000000000000010
map[12]=0b000000010000000

#Table for the type of the nodes
maptype[0]=0b01
maptype[1]=0b01
maptype[2]=0b10
maptype[3]=0b10
maptype[4]=0b11
maptype[5]=0b11
maptype[6]=0b11
maptype[7]=0b11
maptype[8]=0b00
maptype[9]=0b00
maptype[10]=0b00
maptype[11]=0b00
maptype[12]=0b00

#Node type:
#00=location
#01=roundabout
#10=crossroad
#11=branching
#Information on the order of the nodes in an intersection 001 010 011 100
#The bit bit coding represents the nodes connected with 3 bits(8 possibilities of connections) but it is saved on 16 bit there
orientedIntersection[0]=0b011010001
orientedIntersection[1]=0b011010001
orientedIntersection[2]=0b011010100001
orientedIntersection[3]=0b011010100001
orientedIntersection[4]=0b010001
orientedIntersection[5]=0b010001
orientedIntersection[6]=0b010001
orientedIntersection[7]=0b001010

#####

#Turn off some firmware behaviour
call leds.prox.h(0,0,0,0,0,0,0,0)
call leds.prox.v(0,0)
call leds.temperature(0,0)
call prox.comm.enable(1)
#subroutines called by communication

#Stop the Thymio
sub allStop
  if state!=S_STOP then
    oldState=state
    state=S_STOP
    motor.left.target=0
    motor.right.target=0
  end

#Make the Thymio move
sub allStart
  if state==S_STOP then
    state=oldState
    speedType=savedSpeed
  end

#Gives the compatible node in the variable newNode
sub searchCompatibleNode
  nodeTab=[-1,-1,-1,-1,-1]
  nodeCount=0
  stick = map[currentNode]

  #Making of the connected node Tab, it contains all the nodes connected to the currentNode. It find the indice corresponding
  t=0
  oldIndice=-1
  for i in 2:17 do
    if maptype[currentNode] == 00 then
      if stick%0b10 == 0b1 then
        newNode = i-2
        return
      end
    else
      if stick%0b10 == 0b1 then
        nodeTab[t]=i-2
        if i-2==oldNode then
          oldIndice=t
        end
        t++
      end
    end
  end

```

```

    end
    stick = stick >> 1
end

#Treatment of the case of the branching intersection
if maptype[currentNode] == 0b11 then
    if counterContinue==0 then
        newNode=nodeTab[orientedIntersection[currentNode]%0b1000-1]
    else
        newNode=nodeTab[(orientedIntersection[currentNode]>>3)%0b1000-1]
    end
    return
end

#Special case 1 treatment (see section 4.3 in the global part report)
if oldIndice==1 then

    for i in 0:3 do
        if i<t then

            stick = map[nodeTab[i]]
            stick=stick>>(oldNode)
            if stick%0b10 == 0b1 then
                oldIndice=i
            end
        end
    end
    if oldIndice==1 then
        ##If nothing is found, leave the loop
        return
    end
end

stick=orientedIntersection[currentNode]

test2=1
t=0
counNode=0
#Scan the oriented Intersection the old node
while test2==1 do
    if stick%0b1000==(oldIndice+1) then
        test2=0
    end
    counNode++
    stick = stick >> 3
    if counNode>4 then
        return
    end
end

#Compute the exit taken during last intersection
counterContinue=counterContinue+2*countExit
if counterContinue==0 then
    counterContinue=2
end

#San the intersection until reach the right number of exit taken by the robots in the intersection
countExit=1
for i in 0:4 do

    if counNode==5 then
        stick=orientedIntersection[currentNode]
        counNode = 0
    end
    if stick%0b1000!=0b000 then

        if countExit==(counterContinue/NbCont) then

            newNode=nodeTab[stick%0b1000-1]
            return
        end
        countExit++
    end
    counNode++
    stick = stick >> 3
end

###Routine of map management
sub mapUpdate
#Called when there is a modification of node(Edit error detection if two roundabout are next to each other)
callsub searchCompatibleNode

oldNode = currentNode
currentNode = newNode
if (maptype[newNode]%0b100) == newNodeType then
#LEDs to indicate node type
# if currentNode==0 then
## call leds.top(0,32,0)
# elseif currentNode==1 then
## call leds.top(32,0,0)
# else
## call leds.top(0,0,32)
# end
else
#Special case 2 (see section 4.3 in the global part report)
if maptype[newNode]%0b100==0b11 then
    counterContinue=2
    countExit=0
    callsub searchCompatibleNode
    oldNode = currentNode

```



```

        currentNode = newNode
    else
        positionError = 1
    end
end
counterContinue = 0
countExit = 0

#Start the robot in a location before the barcode
onevent button.forward
if button.forward == 1 then
    speedType = SPEED_LOW
    state = S_EXIT
    actionType = A_LOCATION
    readLocation = 1
    decisionState = DS_STOP
    counter = 60
    call prox.comm.enable(1)
end

#Stop the robot
onevent button.center
if button.center == 1 then
    motor.left.target = 0
    motor.right.target = 0
    state = S_STOP
    #comm_state=CS_EAR
    #last_ok =11
    #i=0
    #prox.comm.tx=0
end

onevent prox
#if the robot is not stopped or across another road, it follows the line
if state != S_STOP and state != S_CONTINUE then
    intensityDiff = prox.ground.delta[0]-GREY_REF
    #The robot is given a reference mean value. Its speed is controlled to stay as close as possible
    #to this value with a P controller. The gain (dividing factor) depends on the state.
    if state == S_ROAD or state == S_BACK then
        controlFactor = 3
    elseif state == S_ACTION or state == S_EXIT then
        controlFactor = 2
    elseif state == S_READ then
        controlFactor = 6
    end
    addSpeedLeft = intensityDiff/controlFactor
    addSpeedRight = -intensityDiff/controlFactor

#Obstacle avoidance
if (prox.horizontal[1] > OBS_PROX_THRESH or prox.horizontal[2] > OBS_PROX_THRESH or
    prox.horizontal[3] > OBS_PROX_THRESH) and (state == S_ROAD or state == S_ACTION or state == S_EXIT)
    then
        #Collect the maximum intensity detected
        call math.max(intensityObst, prox.horizontal[2], prox.horizontal[3])
        call math.max(intensityObst, intensityObst, prox.horizontal[1])
        intensityDiffObst = intensityObst - OBS_PROX_THRESH
        #Reduce the speed proportionally to the intensity.
        if intensityDiffObst >= 0 then
            motor.left.target = speedType+addSpeedLeft-intensityDiffObst/(75/(1+speedType/SPEED_LOW))
            motor.right.target = speedType+addSpeedRight-intensityDiffObst/(75/(1+speedType/SPEED_LOW))
        end
        #Above the threshold, stop
        if intensityObst > OBS_PROX_THRESH_MAX then
            motor.left.target = 0
            motor.right.target = 0
        end
    #Obstacle avoidance when going backward. If there is something, stop (no adaptation of speed
    elseif (prox.horizontal[5] > OBS_PROX_THRESH_MAX or prox.horizontal[6] > OBS_PROX_THRESH_MAX) and state ==
        S_BACK then
            motor.left.target = 0
            motor.right.target = 0
    #No obstacle avoidance
    else
        motor.left.target = speedType + addSpeedLeft
        motor.right.target = speedType + addSpeedRight
        #If there is a too big difference, turn on itself (only on white part of gradient)
        if abs intensityDiff > 300 then
            motor.left.target = intensityDiff/4
            motor.right.target = -intensityDiff/4
        end
    end
end
end

#Reading Synchro bit + security to be sure not to read the black side of the road
if state == S_ROAD and prox.ground.delta[1] < GREY_THRESH and prox.ground.delta[0] < white_threshold then
    #If there is already someone reading the barcode, go backward.
    if (prox.horizontal[1] > OBS_PROX_THRESH or prox.horizontal[2] > OBS_PROX_THRESH or
        prox.horizontal[3] > OBS_PROX_THRESH) then
        state = S_BACK
        timer.period[0] = 1200
        speedType = -SPEED_LOW
        timerON = 1
        #Go to the reading state.
    else
        oldState = state
        state = S_READ
        counterRead = 0
        code = 0
    end
end
end

#Reading barcode
if state == S_READ then

```

```

#Interval depending on the speed
interval = 2*SPEED_HIGH/speedType #count between two readings
counterRead++
#First time wait 1,5 interval to ben in the middle of the next bit
if counterRead == 1+(3*interval-1)/2 then
  if prox.ground.delta[1] < GREY_THRESH then
    code +=1
  end
elseif counterRead == 1+(5*interval-1)/2 then
  if prox.ground.delta[1] < GREY_THRESH then
    code +=2
  end
elseif counterRead == 1+(7*interval-1)/2 then
  if prox.ground.delta[1] < GREY_THRESH then
    code +=4
  end
#Wait 1 additional interval to be sure to be across the barcode
elseif counterRead == 1+(9*interval-1)/2 then
  state = oldState
  #Change state corresponding to the code read
  if state == S_ROAD then
    if code == 1 then
      speedType=SPEED_LOW
    elseif code == 2 then
      speedType=SPEED_MED
    elseif code == 3 then
      speedType=SPEED_HIGH
    elseif code == 4 then
      savedSpeed = speedType
      speedType=SPEED_LOW
      state = S_ACTION
      actionType = A_BRANCHING
      newNodeType = 0b11
      callsub mapUpdate
    elseif code == 5 then
      savedSpeed = SPEED_LOW
      speedType=SPEED_LOW
      state = S_ACTION
      actionType = A_CROSSROAD
      call math.rand(random)
      decisionState = random%3
      newNodeType = 0b10
      callsub mapUpdate
    elseif code == 6 then
      savedSpeed = SPEED_LOW
      speedType=SPEED_LOW
      state = S_ACTION
      actionType = A_LOCATION
      newNodeType = 0b00
      callsub mapUpdate
    elseif code == 7 then
      savedSpeed = speedType
      speedType=SPEED_LOW
      state = S_ACTION
      actionType = A_ROUNDABOUT
      newNodeType = 0b01
      callsub mapUpdate
    end

    elseif readLocation == 1 then
      #Actual position used to reset the map in real time
      currentNode=code+7
    end
  end
end
end

#Intersection
if state == S_ACTION then
  #Crossroad
  if actionType == A_CROSSROAD then
    #decisionState =DS_EXIT
    #After 4 Continue, the robot is out and goes back to road state
    #The 4th continue starts directly after the 3rd one
    if counterContinue == 3 then
      state = S_CONTINUE
      counterContinue++
      counter = 0
    elseif counterContinue == 4 then
      state=S_ROAD
      speedType = savedSpeed
    end
  else
    #Random choice when reading bit
    call math.rand(random)
    decisionState = random%2
    #Location and Branching
    #If continue was chosen, return to road state after two readings
    if (actionType == A_BRANCHING or actionType == A_LOCATION) and counterContinue == 2 then
      state=S_ROAD
      speedType = savedSpeed
    #Roundabout
    else
      #Exit is possible only on even number of readings
      if decisionState == DS_EXIT and counterContinue%2 == 0 then
        nbRond++
      else
        decisionState = DS_CONTINUE
      end
    end
  end
end
end

#Reading a bit + security and reading a code in a location

```

```

if (state == S_ACTION or (state == S_EXIT and readLocation == 1)) and prox.ground.delta[1] < GREY_THRESH and prox.ground
if readLocation == 1 then
  oldState = state
  state = S_READ
  counterRead = 0
  code = 0
else
  counter = 0
  #Change state
  if decisionState == DS_EXIT then
    state = S_EXIT
  else
    state = S_CONTINUE
    counterContinue++
    #In case of a crossroad and turning left, if 2 continue done, force exit decision state to follow
    #line
    if (actionType == A_CROSSROAD and decisionState == DS_TURN_LEFT and counterContinue == 2) then
      decisionState = DS_EXIT
    end
  end
end
end

#Exit state
if state == S_EXIT then
  counter++
  #In a location, prepare to stop
  if actionType==A_LOCATION then
    if prox.ground.delta[1] > GREY_THRESH then
      readLocation = 1
    end
    decisionState=DS_STOP
  end
  #Counter condition for other intersections
  if (counter > 41 and (actionType == A_ROUNDABOUT or actionType == A_BRANCHING)) or (counter > 29 and actionType == A_CROSSROAD) then
    #If decision was to turn left, use exit state to follow line, then goes to action state (and not road)
    countExit++
    if actionType == A_CROSSROAD and counterContinue != 0 then
      state = S_ACTION
      decisionState = DS_TURN_LEFT
    else
      #Goes back to road state except for location intersection
      if decisionState!=DS_STOP then
        state = S_ROAD
        speedType = savedSpeed
      end
    end
  end
end

#Cross a road or cross a part without gradient
if state == S_CONTINUE then
  counter++
  #Due to placement after a left turn, the robot is never going straight. Therefore add a bias to make it
  #turn slightly to the left, depends on intersection
  motor.left.target = (85+actionType*3)*speedType/100
  motor.right.target = speedType

  #Obstacle avoidance. If there is something, stop counter and stop moving
  if (prox.horizontal[1] > OBS_PROX_THRESH or prox.horizontal[2] > OBS_PROX_THRESH or
    prox.horizontal[3] > OBS_PROX_THRESH) then
    savedSpeedInters = speedType
    motor.left.target = 0
    motor.right.target = 0
    counter--
    counterStop = 1
  #No more obstacles
  elseif counterStop == 1 and (prox.horizontal[1] < OBS_PROX_THRESH and prox.horizontal[2] < OBS_PROX_THRESH
    and prox.horizontal[3] < OBS_PROX_THRESH) then
    counterStop = 0
    speedType = savedSpeedInters
  end

  #Goes back to action state
  if counter > 21 then
    state = S_ACTION
  end
end

#Location stop
if decisionState == DS_STOP then
  if ((prox.horizontal[1] > OBS_PROX_THRESH_MAX-100 or prox.horizontal[2] > OBS_PROX_THRESH_MAX-100 or
    prox.horizontal[3] > OBS_PROX_THRESH_MAX-100)) then
    counter--
  end
  if counter > 130 then
    #Increment only if traffic light is green
    counter=counter + incrementCounter
    state = S_STOP
    motor.left.target = 0
    motor.right.target = 0
    #After a while goes back to road state
    if counter > 150 then
      decisionState = DS_NOTHING
      speedType = savedSpeed
      motor.left.target = speedType
      motor.right.target = speedType
      readLocation = 0
      state = S_ROAD
    end
  end
end
end

```

```

#LEDs to indicate state
if state == S_ROAD then
  call leds.top(32,(3-((speedType-SPEED_LOW)/SPEED_LOW))*8,0)
elseif state == S_READ then
  call leds.top(32,32,32)
elseif state == S_ACTION then
  call leds.top(0,10*actionType,10*(3-actionType))
end

#Communication
onevent prox.comm
# Bidirectionnal communication
# if comm_state==CS_TELL then
#   if prox.comm.rx==last_ok then
#     #call leds.top(0,0,32)
#     if last_ok==10 then
#       last_ok=11
#     else
#       last_ok=10
#     end
#     if i==1 then
#       prox.comm.tx=currentNode+60
#     elseif i==2 then
#       prox.comm.tx=oldNode+60
#     else
#       prox.comm.tx=currentNode + 100
#     end
#     timer.period[1]=2000
#     comm_state=CS_EAR
#     last_ok =11
#   end
#   count++
#   i++
# end
# end
if comm_state==CS_EAR then
  if maptype[currentNode]==0b10 then
    if prox.comm.rx==oldNode+20 and counterContinue==0 and countExit == 0 then
      callsub allStop
    else
      if prox.comm.rx==oldNode+40 then
        callsub allStart
      end
    end
  end
  if counter < 150 then
    if maptype[currentNode]==0b00 and counterContinue == 0 and prox.comm.rx == currentNode+20 then
      callsub allStart
      incrementCounter = 1
    elseif maptype[currentNode]==0b00 and counterContinue == 1 and prox.comm.rx == currentNode+20 then
      callsub allStop
      incrementCounter = 0
    elseif maptype[currentNode]%0b100==0b00 and counterContinue==0 and prox.comm.rx == currentNode+40 then
      callsub allStop
      incrementCounter = 0
    elseif maptype[currentNode]%0b100==0b00 and counterContinue==1 and prox.comm.rx == currentNode+40 then
      callsub allStart
      incrementCounter = 1
    end
  end
end
# Code for the communication state machine
# elseif prox.comm.rx==1 then
#   comm_state=CS_TELL
#   prox.comm.tx=3
#   i=0
# Code for the Ping part
# elseif prox.comm.rx>100 then
#   timer.period[1]=2000
#   if countPing==1 then
#     prox.comm.tx=prox.comm.rx+20
#     countPing=2
#   else
#     prox.comm.tx=prox.comm.rx
#   end
# end

end

#Timer used to go backward
onevent timer0
  if timerON == 1 then
    speedType = SPEED_LOW
    state = S_ROAD
    timerON = 0
  end

#Timer used for communication
onevent timer1
  if comm_state==CS_EAR then
    prox.comm.tx=0
    timer.period[1]=0
  end
  countPing = 1

</node>

</network>

```

C Code Thymio signaux

```

<!DOCTYPE aesl-source>
<network>

<!-- list of global events -->
<event size="0" name="Reception"/>
<event size="1" name="reception2"/>
<event size="1" name="slavecarrefour2"/>
<event size="1" name="slavecarrefour3"/>

<!-- list of constants -->
<constant value="0" name="ear_sc"/>
<constant value="1" name="tell_sc"/>
<constant value="20" name="max_comm"/>
<constant value="2" name="ask_sc"/>

<!-- show keywords state -->
<keywords flag="true"/>

<!-- node thymio-II -->
<node nodeId="1" name="thymio-II">
#Communication storage tab
var communication[max_comm]=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
var indexComm = 0
var indexDebut=0

#State variable
var state_comm=tell_sc
var last_ok =1
var i=0
var thymio_pos[2]
var thymio_bef[2]
var index_thymio=0
var envoie
var count=0
var compteurAsk=0

#phase of the road controlled by the signal
#0=all stop
#1=green 1
#2=green 2
#3=green 3
#4=green 4
#Node connected to the cross-road controled by the signal
var pos[4]=[4,3,7,2]
var signal=1
var tab_tell[1]

var compteur
var green=1

call prox.comm.enable(1)
timer.period[0]=300

# sub for the communication
sub gestionComm
  for i in 0:19 do
    if communication[i]==3 then
      indexDebut=i
    end

    end
    thymio_pos[index_thymio]=communication[indexDebut+1]-60
    thymio_bef[index_thymio]=communication[indexDebut+2]-60
    state_comm=tell_sc
#sub for the broadcastdisc
sub tell
  signal++
  count++
  if signal>4 then
    signal=1
  end

  if signal==green then
    if signal==1 then
      call leds.circle(1,0,0,0,0,0,0,0)
    elseif signal==2 then
      call leds.circle(0,0,1,0,0,0,0,0)
    elseif signal==3 then
      call leds.circle(0,0,0,0,1,0,0,0)
    elseif signal==4 then
      call leds.circle(0,0,0,0,0,0,1,0)
    end
    if count>10 then
      envoie=pos[signal-1]+40
    else
      envoie=pos[signal-1]+20
    end
  else
    #all phases red time
    envoie=pos[signal-1]+20
  end

  prox.comm.tx=envoie
#event to use the slave robot to send the broadcast disc
emit slavecarrefour3(envoie)

```

```

onevent button.forward == 1 then
  call prox.comm.enable(1)
  timer.period[0]=400
end

onevent button.center
if button.center == 1 then
  call prox.comm.enable(0)
  indexComm = 0
  indexDebut=0
  state_comm=tell_sc
  last_ok =1
  i=0
end

onevent button.left
green = 4

onevent button.right
green = 1

onevent button.backward
if button.backward==1 then
  call leds.top(0,0,0)
  call leds.circle(0,0,0,0,0,0,0,0)
  prox.comm.tx=0b0
  call prox.comm.enable(0)
end

onevent timer0
compteur++
#time contrôle of the lights
if compteur>50 then
  compteur=0
  green++
  count=0
  if green>4 then
    green=1
  end
end
if state_comm==ask_sc then
  compteurAsk++
  prox.comm.tx=1
  if compteurAsk>50 then
    state_comm=tell_sc
  end
end
if state_comm==tell_sc then
  callsub tell
end

#Instruction#
#code==
#1==Bonjour#
#3==Début#
#4==Fin#
#5==Position#
#6==Etat#
#7==Stop#
#8==Start#
#9==Voie libre#
#10==ok1#
#11==ok2#
#position =x -1000

onevent prox.comm
#state machine
if state_comm==tell_sc then
  if prox.comm.rx==1 then
    state_comm=ask_sc
    compteurAsk=0
    prox.comm.tx=1
  end
end
if state_comm==ask_sc then
  if prox.comm.rx==3 then
    state_comm=ear_sc
    indexComm=0
    prox.comm.tx=11
    last_ok=2
    communication[indexComm]=prox.comm.rx

  end
end
if state_comm==ear_sc then
  if communication[indexComm]!=prox.comm.rx then
    indexComm++
    if indexComm == max_comm then
      indexComm=0
    end
    communication[indexComm]=prox.comm.rx
    if last_ok == 1 then
      prox.comm.tx=11
      last_ok = 2
    else
      prox.comm.tx=10
      last_ok = 1
    end
  end
  if prox.comm.rx==4 then
    callsub gestionComm

```

```
        elseif prox.comm.rx==7 then
        elseif prox.comm.rx==8 then
        end
    end
end</node>

</network>
```

D Code signal de parking

```

<!DOCTYPE aesl-source>
<network>

<!-- list of global events -->
<event size="0" name="Reception"/>
<event size="1" name="reception2"/>

<!-- list of constants -->
<constant value="0" name="ear_sc"/>
<constant value="1" name="tell_sc"/>
<constant value="20" name="max_comm"/>
<constant value="2" name="ask_sc"/>

<!-- show keywords state -->
<keywords flag="true"/>

<!-- node thymio-II -->
<node nodeId="1" name="thymio-II">var communication [max_comm]=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
var indexComm = 0
var indexDebut=0
var state_comm=tell_sc
var last_ok =1
var i=0
var thymio_pos[2]
var thymio_bef[2]
var index_thymio=0
#0=all stop
#1=green 1
#2=green 2
#3=green 3
#4=green 4
var pos[1]=[8]
var signal=0
var tab_tell[1]
var compteur=5
var green=1
var envoie

sub gestionComm
  for i in 0:19 do
    if communication[i]==3 then
      indexDebut=i
    end

    end
    thymio_pos[index_thymio]=communication[indexDebut+1]-1000
    thymio_bef[index_thymio]=communication[indexDebut+2]-1000
    state_comm=tell_sc

sub tell
  signal++
  if signal>2 then
    signal=1
  end

  if signal==1 then
    call leds.circle(0,0,1,0,0,0,0,0)
    call leds.top(12,0,12)
    envoie=pos[0]+40
  elseif signal==2 then
    call leds.circle(0,0,0,0,0,0,1,0)
    call leds.top(0,12,12)
    envoie=pos[0]+20
  elseif signal==3 then
    call leds.circle(1,1,1,0,0,0,0,0)
  elseif signal==4 then
    call leds.circle(1,1,1,1,0,1,0,0)
  end
  #prox.comm.tx=pos[signal-1]+20
  #call leds.top(0,10,0)

  #call leds.top(10,0,0)

sub allStart

onevent buttons
  if button.forward == 1 then
    call prox.comm.enable(1)
    timer.period[0]=500
  end

onevent button.center
  if button.center == 1 then
    call prox.comm.enable(0)
    indexComm = 0
    indexDebut=0
    state_comm=tell_sc
    last_ok =1
    i=0

```



```

end

onevent button.backward
  if button.backward==1 then
    call leds.top(0,0,0)
    call leds.circle(0,0,0,0,0,0,0,0)
    prox.comm.tx=0b0
    call prox.comm.enable(0)
  end
onevent button.left
  callsub tell

onevent timer0
  compteur++
  if compteur>75 then
    compteur=0
    if state_comm==tell_sc then
      callsub tell
    end
  end
  prox.comm.tx=envoie
  if state_comm==ask_sc then
    prox.comm.tx=1
  end
  if state_comm==tell_sc then
    #callsub tell
  end
#Instruction#
#code==
#1==Bonjour#
#3==Début#
#4==Fin#
#5==Position#
#6==Etat#
#7==Stop#
#8==Start#
#9==Voie libre#
#10==ok1#
#11==ok2#
#position =x -1000

onevent prox.comm
  if state_comm==tell_sc then
    if prox.comm.rx==1 then
      state_comm=ask_sc
      prox.comm.tx=1
    end
  end
  if state_comm==ask_sc then
    if prox.comm.rx==3 then
      state_comm=ear_sc
      indexComm=0
      prox.comm.tx=11
      last_ok=2
      communication[indexComm]=prox.comm.rx
    end
  end
  if state_comm==ear_sc then
    if communication[indexComm]!=prox.comm.rx then
      indexComm++
      if indexComm == max_comm then
        indexComm=0
      end
      communication[indexComm]=prox.comm.rx
      if last_ok == 1 then
        prox.comm.tx=11
        last_ok = 2
      else
        prox.comm.tx=10
        last_ok = 1
      end
      if prox.comm.rx==4 then
        callsub gestionComm
      elseif prox.comm.rx==7 then

      elseif prox.comm.rx==8 then

      end
    end
  end
end</node>

</network>

```