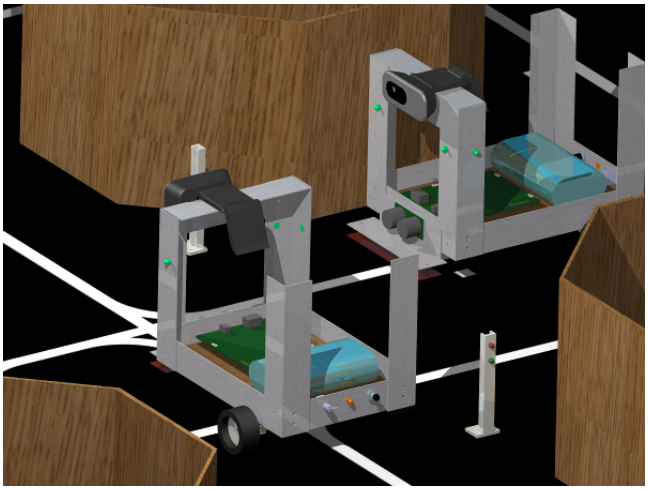
Projet Electif

Projet technique

8 juin 2016

Evolution de véhicules autonomes dans un environnement urbain intelligent



Auteurs :

Biton Guillaume (guillaume.biton@ipsa.fr)

Guichard Marc-Antoine (marc-antoine.guichard@ipsa.fr)

Lhermite Camille (camille.lhermite@ipsa.fr)

Monnot Maxime (maxime.monnot@ipsa.fr)

Table des matières

Remerciements

Nous remercions Mesdames Assia Belbachir et Sorore Benabid, ensignants chercheurs au sein du laboratoire Mécatronique, Signal et Systèmes de l'IPSA pour nous avoir soumis cette idée, pour nous avoir accordé leur confiance quant à sa réalisation potentielle et pour leur soutien dans le cadre de ce projet.

Notes préliminaires

L'ensemble des illustrations présentes dans ce dossier, sauf indication contraire dans leurs légendes, sont des réalisations personnelles et ne sont donc soumises à aucun droit particulier.

Le travail de recherche ici présenté est sous la responsabilité des auteurs cités en première de couverture et sous la propriété de l'IPSA. Merci de nous contacter au moyen des adresses fournies pour nous signaler tout problème, ou pour toute demande liée à l'utilisation du contenu.

L'ensemble des abréviations utilisées dans ces pages sont explicitées en page 32 (nomenclature).

Les numéros entre crochets pouvant apparaitre au sein de paragraphes sont des références bibliographiques et vous renvoient à la page 33 (bibliographie).

Ce dossier comprend de nombreuses références dynamiques, et pour un confort de lecture optimal, nous vous conseillons d'en lire la version numérique afin de pouvoir bénéficier des liens cliquables.

Exception faite des modèles mécaniques en CAO 3D effectués sous Catia®, l'ensemble des réalisations ici présentées utilisent des outils Open-Source : le présent dossier est rédigé en LaTeX, les illustrations ont été réalisées en dessin-vectoriel sous Inkscape ou en matriciel sous GIMP et les schémas et circuits électroniques sous KiCad.

1 Introduction

La voiture autonome, loin du concept de science-fiction qu'elle pouvait représenter il y a quelques années est en train de devenir une réalité.

Si cette ambition put être à une certaine époque motivée par le simple attrait de la prouesse technique, nous percevons aujourd'hui tous les bénéfices que l'on pourrait en tirer. En effet, les avancées scientifiques et techniques nous permettent désormais de prétendre à concevoir une voiture qui soit non seulement autonome, mais surtout intelligente. Il est aujourd'hui tout à fait réaliste de penser que dans les quelques années à venir les voitures sauront adopter un comportement bien plus intelligent que celui de leurs conducteurs actuels, et ce au profit de la sécurité, de l' efficience énergétique mais également de l'encombrement des axes routiers.

A terme, nous pouvons facilement imaginer que les différents véhicules auront la possibilité de communiquer entre eux afin de prévenir les véhicules environnants de leurs intentions, mais cela ne les dispensera pas de devoir être capables d'évaluer leur environnement afin d'y détecter les éléments "indépendants" (piétons, obstacles...).

Comme toute révolution technologique, la voiture intelligente devra faire face au caractère progressif de son adoption : toutes les voitures sur les routes ne deviendront pas autonomes du jour au lendemain. Ces véhicules devront donc également être capables d'évoluer au milieu d'une circulation telle que nous la connaissons, où chaque acteur adopte un comportement presque parfaitement aléatoire, et ne signale pas toujours ces intentions.

Si les voitures deviennent intelligentes, c'est également le cas, depuis quelques années déjà, de la signalisation. Ainsi, de plus en plus de feux adaptent leur comportement au trafic, ce qui s'inscrit également dans une démarche d'optimisation de la circulation. Il est très réaliste d'espérer que cette technologie déjà existante se fera omniprésente dans les années qui viennent, d'où notre volonté d'inclure cet élément d'environnement à notre projet.

Le but de ce projet est donc d'étudier notre capacité à faire cohabiter intelligence artificielle et environnement "réel" et indépendant avec des moyens techniques et financiers extrêmement restreints, mais également et surtout de fournir une plateforme d'expérimentation aux étudiants de l'IPSA (et d'ailleurs ?) afin de faciliter l'apprentissage des sciences de la mécatronique et de l'intelligence artificielle tout en ancrant ce savoir dans une réalité pratique et stimulante.

2 Définition du besoin

Une définition pertinente du besoin est une condition absolument nécessaire pour la bonne réalisation de tout projet. Nous nous emploierons donc à définir aussi précisément et pertinemment que possible le besoin motivant ce projet, et à régulièrement revenir sur ce dernier afin de prendre en compte d'éventuelles évolutions et à prévenir toute dérive du projet.

2.1 Besoin

Le besoin à l'origine de ce projet fut exprimé par mesdames A. Belbachir et S. Benabid en tant qu'enseignants chercheurs au laboratoire Mécatronique, Signal et Systèmes à l'IPSA. Leur souhait était de bénéficier d'une plateforme articulée autour de robots autonomes évoluant dans une simulation d'environnement urbain, capables de détecter des feux de signalisation "intelligents" et d'adapter leur comportement en conséquence. Cette plateforme pourrait servir de support de TP dans l'ensemble des matières enseignées par le département, mais également servir de "vitrine" voir même de plateforme de recherche.

Les principales fonctionnalités évoquées étant :
- **La reconnaissance d'image** pour la détection des feux de signalisation.
- **La présence de feux bicolores commandés par FPGA.** Ces feux seraient "intelligents" dans le sens où il s'adapteraient à la circulation. La présence de capteurs de présence est donc induite.

2.2 Précision du besoin

Ayant suivi nombre de matières enseignées par le département et participé à de nombreuses séances de travaux pratiques, nous bénéficions d'une image relativement claire des implications de ce projet ainsi que des contraintes relevant pour la plupart du simple bon sens. Dans un soucis de clarté et d'application de "bonnes pratiques" nous prîmes cependant soin de valider l'ensemble de ces éléments au cours de réunions avec nos "clients" du laboratoire.

Ainsi, nous ajoutâmes les points suivants à la liste des exigences :
- "Côté circuit" :
  - **Le circuit devra bénéficier d'un encombrement raisonnable**.
  - **Les feux de circulation devront pouvoir être commandés via une carte FPGA**.
- "Côté robots" :
  - **Les robots devront pouvoir être utilisés en classe sans que les préoccupations matérielles ne soient accaparantes**.
  - **Les robots devront pouvoir être programmés en utilisant les langages enseignés à l'IPSA** à savoir C et C++ et ses variantes (Arduino...), Python, voir même Matlab...

— **Les robots devront bénéficier de possibilités d'application flexibles :** les enseignements étant ciblés, il est important que les utilisateurs des robots puissent se concentrer sur un aspect de leur utilisation sans avoir à se soucier des autres. De même, les robots devront embarquer suffisamment de capteurs ou tout du moins de capacité d'extensions pour que cela ne soit pas un facteur limitant lors de l'élaboration de sujets de TP.
— **Les robots devront ne pas pouvoir représenter un danger pour ses utilisateurs**.

3 Étude du besoin

3.1 Analyse du besoin

Le besoin exprimé précédemment comporte un certain nombre d'implications logiques. Leur prise en compte relève déjà du domaine de la solution, mais reste d'ordre suffisamment général pour être évoquée ici. Surtout, ces points seront à prendre en compte lors de l'élaboration d'une solution "détaillée".

Reprenons les points évoqués ci-avant :

3.1.1 Pour le circuit

- **Le circuit devra bénéficier d'un encombrement raisonnable**.
  ⇒ Les dimensions du circuit devront être raisonnables.
  ⇒ Nous songerons également à une structure démontable (qui devra rester simple à assembler).
- **Les feux de circulation devront pouvoir être commandés via une carte FPGA**.
  ⇒ Une carte adaptant les tensions et courants et disposant de connecteurs adaptés aux caractéristiques des cartes possédées par l'IPSA devra être proposée.

3.1.2 Pour le robot

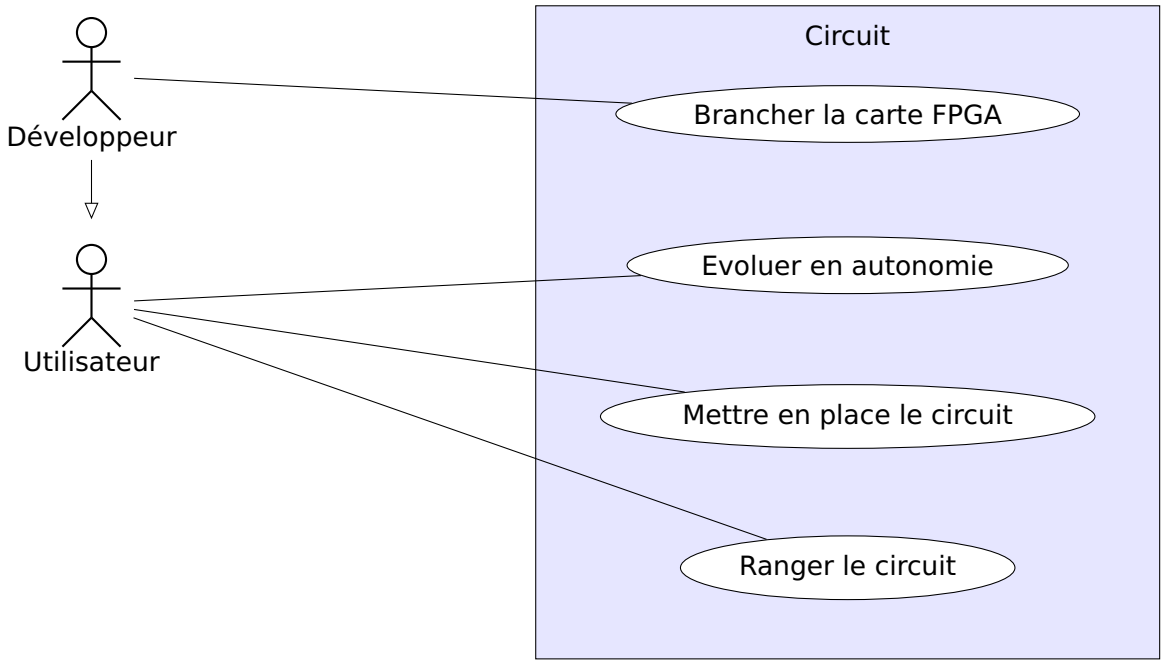
- **Les robots devront pouvoir être utilisés en classe sans que les préoccupations matérielles ne soient accaparantes.**
  ⇒ Cela implique plusieur points :
  - Bénéficier d'une autonomie suffisante pour que l'ensemble des groupes puissent effectuer leurs manipulations au cours d'une séance entière sans que cela ne nécéssite de charger/remplacer la batterie. D'expérience, nous estimons l'autonomie minimum à une heure pour répondre à ce critère.
  - Présenter des connecteurs et des interfaces permettant une intéraction "technique" avec le robot ne nécessitant pas de matériel complexe. Ceci passe par la possibilité de changer ou recharger la batterie sans outils particuliers, de pouvoir programmer le robot via USB ou réseau plutôt que via un port parallèle par exemple, ou encore proposer au minimum un bouton accessible permettant le lancement d'un séquence de code prédéfinie.
  - Etre d'une conception suffisamment robuste pour pouvoir être manipulé quotidiennement sans en "souffrir" et suffisamment simple pour pouvoir être aisément entretenu.
- **Les robots devront pouvoir être programmés en utilisant les langages enseignés à l'IPSA.**
  ⇒ Cette problématique sera étudiée plus en détail en 4.2.3.2 (page 17)

- **Les robots devront bénéficier de possibilités d'application flexibles.**
  ⇒ Ceci passe par une **modularité totale** des différents éléments qui composent le robot, aussi bien sur le plan matériel que logiciel. Cette modularité devra rester une préoccupation de premier plan tout au long du développement de ce projet.
- **Les robots devront ne pas représenter de danger pour ses utilisateurs**.
  ⇒ La structure du robot ne devra pas présenter d'éléments tranchants.
  ⇒ La partie électrique et électronique devra employer des puissances raisonnables, et être protégée contre les forts courants.

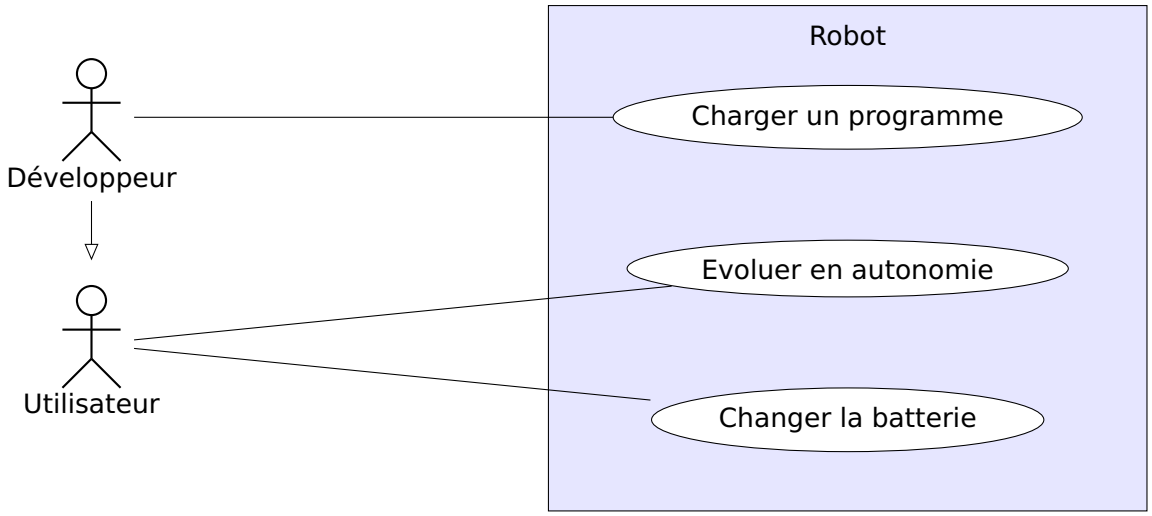
3.2 Diagrammes des cas d'utilisation

Toujours dans un soucis de clareté et de bonne pratiques, nous avons ébauché les diagrammes des cas d'utilisation de haut niveau suivants :

3.2.1 Pour le circuit



3.2.2 Pour le robot



Reprogrammation :

- **Précondition :** Nous disposons d'un programme fonctionnel.
- **Déclencheur :** Un développeur souhaite implémenter un programme.
- **Scenario :**

1. On connecte le robot à une source de tension.
2. On établit une liaison entre le robot et un ordinateur.
3. Le développeur lance le téléversement du programme.
4. Le robot acquitte.
5. On ferme la connexion.

Evolution en autonomie :

- **Précondition :** Le robot a été programmé et dispose d'une batterie chargée.
- **Déclencheur :** Un membre du laboratoire souhaite observer le comportement du robot.
- **Scenario :**

1. On met le robot sous tension.
2. On place le robot sur une ligne blanche du circuit.
3. On appuie sur le bouton de démarrage de séquence
4. Le robot déclenche les programmes en mémoire.

Changement de batterie :

- **Déclencheur :** On souhaite changer la batterie du robot
- **Scenario :**

1. On met le robot hors tension.
2. On accède à la batterie en place le cas échéant, et on la retire.
3. On met en place la nouvelle batterie
4. On remet en place les éléments éventuellement retirés pour accéder à la batterie.

Le mode d’utilisation primaire est bien évidemment celui de l’évolution en autonomie. Le robot étant destiné à servir de plateforme de recherche, et devant donc être entièrement reprogrammable, il est délicat de décrire ce mode d’utilisation qui dépendra intégralement du programme chargé par l’utilisateur.

Nous nous appliquerons cependant à décrire le mode d’utilisation correspondant à l’application la plus basique du robot (et de fait au programme que nous livrerons avec ce dernier).

4 Élaboration d’une solution

4.1 Circuit

4.2 Robot

4.2.1 Suivi de trajectoire

Afin d’offrir une capacité de suivi de trajectoire simple et robuste, il a été décidé d’intégrer au robot la fonctionnalité "suivi de ligne". Il s’agit de l’une des méthodes les plus répandues[1] [7], relativement simple à implémenter et économe aussi bien en composants qu’en puissance de calcul nécessaire.

L’idée est d’offrir une base fiable et simple pour que le suivi de trajectoire ne soit pas une source de préoccupation ou d’erreur dans le cas d’utilisations centrées sur d’autres problématiques (dans le cadre d’un TP sur la reconnaissance d’image, par exemple). Cela n’exclut cependant pas qu’un étudiant ou chercheur désireux d’explorer d’autres possibilités de suivi de trajectoire (via la caméra, un dispositif de triangulation ou autre) puisse se passer de ce module et exploiter une autre solution.

Le principe de suivi de ligne est relativement simple : on place sur l’axe du robot, quelques millimètres au-dessus du sol, un capteur appelé « réflecteur optique ». Ce capteur émet une onde lumineuse (souvent infrarouge) et une cellule mesure l’intensité reçue sur la longueur d’onde émise. Une forte intensité reçue indiquera la présence d’une surface réfléchissante, tandis qu’une faible intensité indiquera la présence d’une surface absorbante. Il est ainsi aisé de différencier un fond sombre (la « route ») d’une ligne blanche.

Une loi linéaire lie la tension lue en sortie de capteur à l’intensité reçue.
Une simple lecture de cette tension permet, après comparaison avec des valeurs "seuil" définies expérimentalement, de savoir si le capteur se trouve au dessus d’une ligne blanche ou non.

Sur les schémas suivants, pour des raisons d’imprimabilité et de lisibilité, les lignes seront noires et le revêtement du circuit sera blanc.

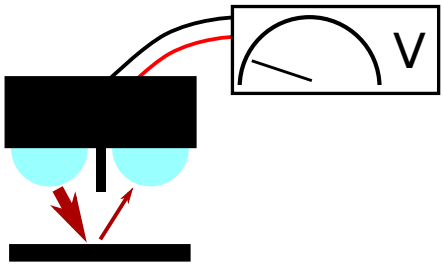


Figure 1 – Capteur au dessus d’un support sombre

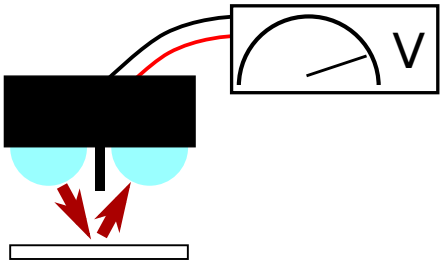


Figure 2 – Capteur au dessus d’un support clair

La question qui se pose est celle du nombre de capteurs, et de leur disposition.

Il est tout à fait possible de n'utiliser qu'un capteur : chaque fois qu'il quitte la ligne blanche, on entamera un virage à droite (puis à gauche si on ne retrouve pas la ligne blanche dans les quelques millisecondes suivantes) jusqu'à retrouver la ligne. Il est évident que cette méthode ne permettra pas une très grande fluidité de déplacement pour notre robot.

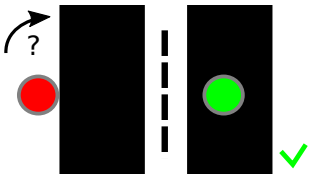


Figure 3 – Dépassement de la ligne sur la gauche

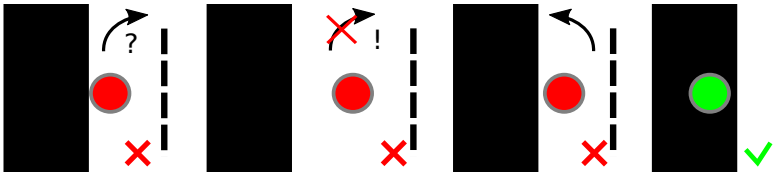


Figure 4 – Dépassement de la ligne sur la droite

L'utilisation de deux capteurs permet une meilleure fluidité. On placera cette fois un capteur de chaque côté de la ligne.

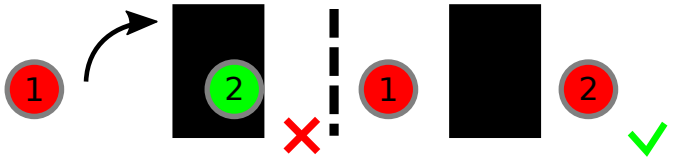


Figure 5 – Dépassement de la ligne sur la gauche

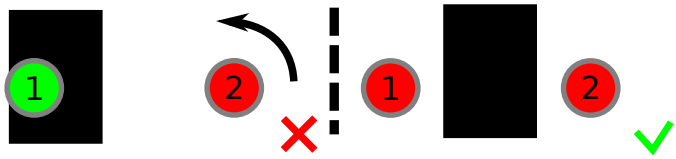


Figure 6 – Dépassement de la ligne sur la droite

Notons que, quelque soit la méthode employée, le suivi de ligne se résume toujours à un "rebondissement" des capteurs sur la ligne. Aussi, pour obtenir une trajectoire aussi rectiligne que possible, il faudra "resserer" les capteurs au maximum autour de la ligne, et appliquer des corrections de faible amplitude.

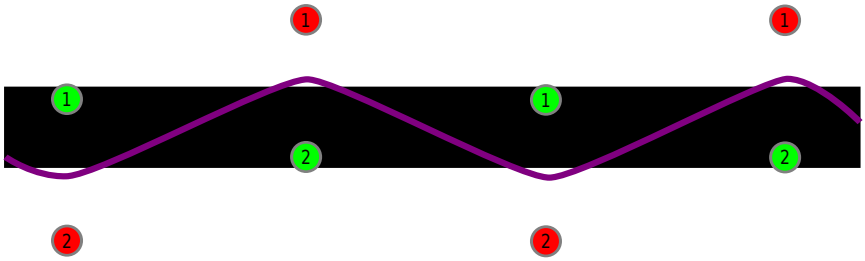


Figure 7 – Ecart important entre les capteurs

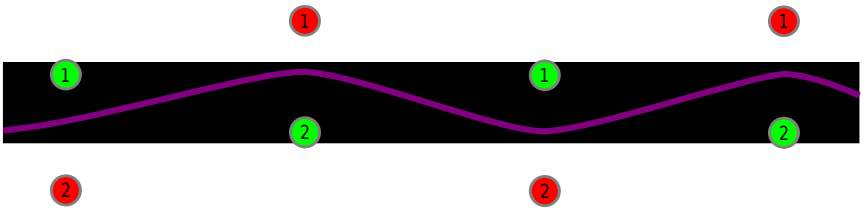


Figure 8 – Ecart réduit entre les capteurs

Nous pourrons éventuellement introduire un amortissement progressif de la correction via un régulateur PID, mais cela peut introduire de nombreuses problématiques dans des cas d'utilisation plus complexes.

Nous retiendrons donc à ce stade la solution consistant en deux capteurs placés de part et d'autre de la ligne. Nous veillerons à ce que le placement des capteurs permette une détection de très faibles écarts de trajectoire, sans pour autant induire des ambiguïtés de mesure.

Intéressons nous maintenant au cas d'une trajectoire courbe. Le fait d'avoir privilégié l'application de faibles corrections peut alors s'avérer problématique si ces dernières ne sont pas suffisantes pour adapter la trajectoire.

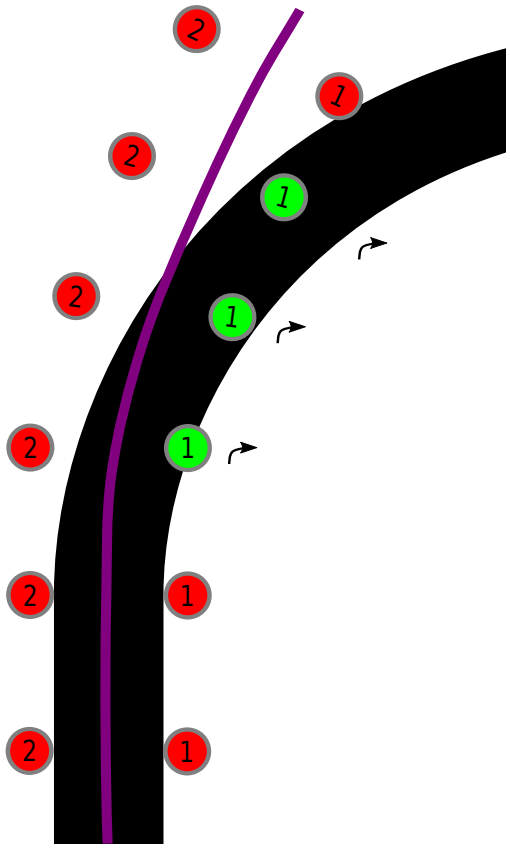


Figure 9 – Perte de trajectoire en virage dû à de trop faibles corrections

Pour obtenir un fluidité de déplacement raisonnable tout en assurant la possibilité de suivre des trajectoires courbes à rayon de virage réaliste, nous placerons une deuxième série de capteurs "encadrant" les capteurs principaux. Une détection de ligne par ces capteurs déclanchera une correction plus importante.

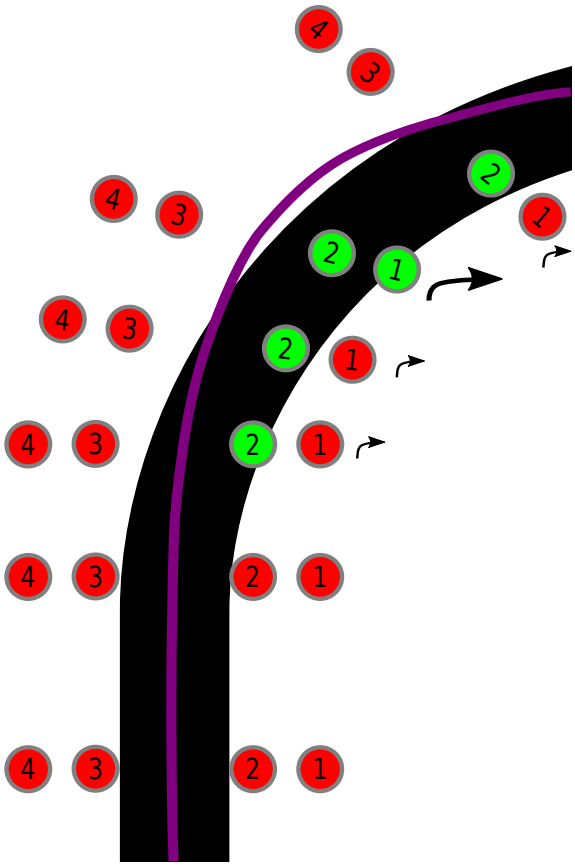


Figure 10 – Maîtrise de la trajectoire en virage grâce à l'utilisation de 4 capteurs

Cette solution consistant en l'utilisation de quatre réflecteurs optiques alignés de manière à encadrer de près la ligne à suivre est sans doute l'une des plus simples imaginables en terme de réalisation mais également d'utilisation et de maintenance. Elle n'est malheureusement pas suffisante pour répondre à notre besoin. En effet, cette solution trouvera ses limites dès lors que notre Robot rencontrera sa première intersection, élément capital de ce projet.

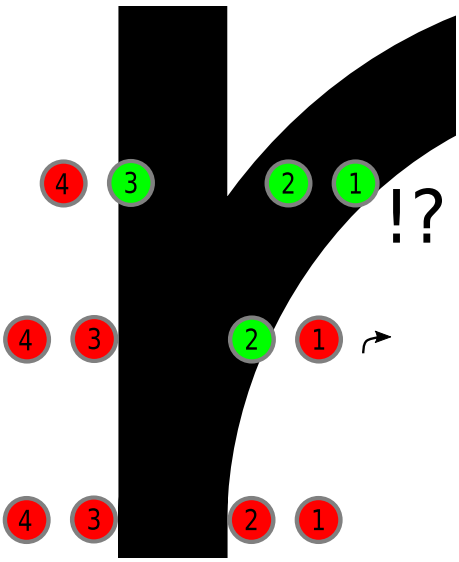


Figure 11 – Robot incapable de répondre à la présence d'une intersection

Si on trouve dans la littérature de très nombreux exemples de réalisations de robots suiveurs de lignes, ces dernières se limitent toujours à des circuits sous forme de boucle. Notre défi est donc de mettre au point une solution permettant l'évolution du robot sur un circuit comprenant des intersections. Notre robot devra ne pas être "perturbé" au passage d'une intersection, mais surtout être capable d'emprunter toutes les directions qui lui sont offertes.

La solution que nous avons mis au point consiste en l'introduction d'un capteur central et l'adoption d'un comportement "mono-latéralement centré" du robot lors de la traversée d'intersections : lorsqu'il arrivera sur une intersection, le robot ne se souciera plus que de garder son capteur central sur la ligne, et ses capteurs latéraux **du côté de la direction choisie** à l'écart de cette dernière.

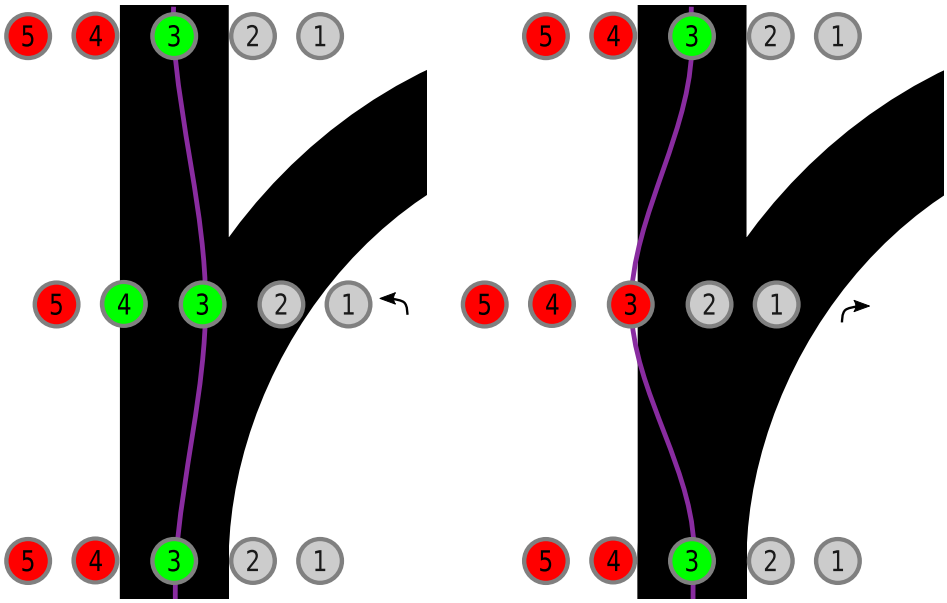


Figure 12 – Suivi de trajectoire sur une intersection (en choisissant d'aller tout droit)

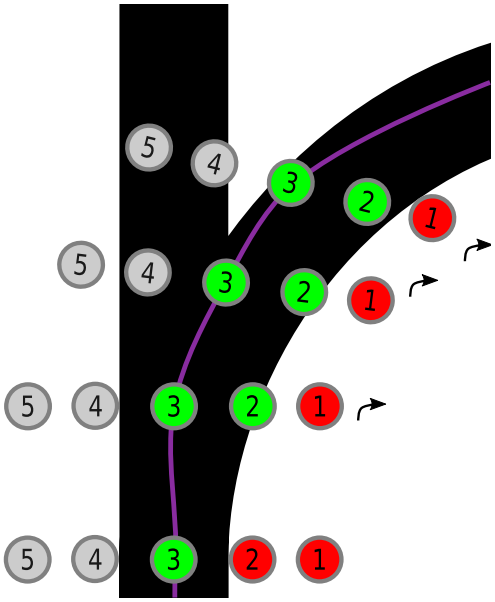


Figure 13 – Suivi de trajectoire sur une intersection (en choisissant d'aller à droite)

Notons que la forme de la ligne devra être intelligemment faite, en présentant des rayons de courbure réalistes (un véhicule ne peut pas tourner à angle droit) et des intersections réalisables (notamment via l'utilisation de deux lignes séparées pour la circulation en sens contraires).

Voici donc à quoi ressemblerait un carrefour :

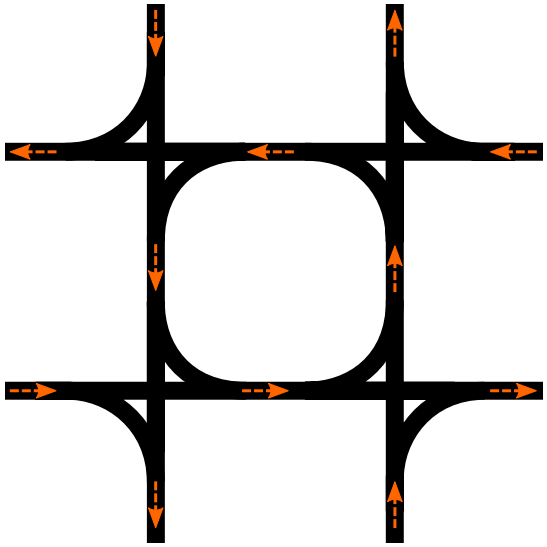


Figure 14 – Carrefour en représentation "lignes"

Pour pleinement définir la logique mise au point, illustrons la au travers de la situation la plus complexe que pourra rencontrer notre robot, qui est celle de la bifurcation à gauche.

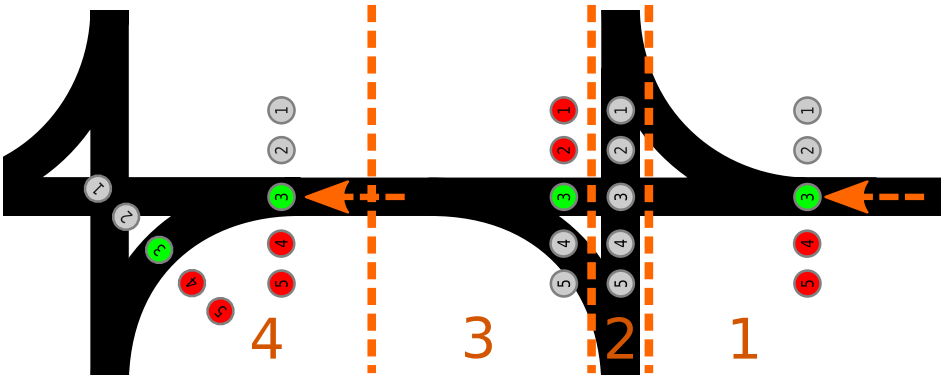


Figure 15 – Cas du virage à gauche sur un carrefour avec cinq capteurs

1. Le premier croisement ne nous intéresse pas. Nous allons continuer tout droit et éteignons donc la moitié droite de nos capteurs.
2. Au passage du croisement, nous éteignons temporairement l'ensemble de nos capteurs.
3. Le croisement passé, nous éteignons cette fois la moitié gauche de nos capteurs pour ne pas être perturbés par "la voie d'insertion".
4. Peu de temps après, nous rallumons la moitié gauche et éteignons la moitié droite des capteurs pour suivre la voie bifurquant à gauche.

Cette solution semble robuste tout en conservant une certaine simplicité d'implémentation. Elle sous-entend en revanche que le robot "sache" quand il arrive sur une intersection, mais également qu'il puisse évaluer à quelques millimètres près la distance qui le sépare des différentes "parties" de cette intersection (pour activer et désactiver ses capteurs en conséquence).

La solution à cette problématique nous a été inspirée par le souvenir d'un brevet déposé en 2003 par le groupe PSA [8], qui proposait l'utilisation de code-barres tracés sur la route et que des capteurs placés sous le pare-choc des voitures pourraient "lire" pour prévenir le conducteur des sources de danger à venir (intersections, feux...). Il fut donc décidé que deux capteurs supplémentaires seraient ajoutés sur les extérieurs. Ils augmenteraient ainsi la précision de placement sur la ligne, mais permettraient surtout de lire des "codes-barres" placés sur la piste. Ces code-barres devraient permettre au robot d'être averti de l'approche d'un carrefour (trois choix de direction possibles) ou d'une simple intersection (deux choix possibles). Nous avons simplement basé notre code sur une ligne simple ou doublée :

	Gauche	**Droite**	**Gauche ET Droite**
Simple	Impossible de tourner à droite	Impossible de tourner à gauche	Toutes possibilités offertes
Double	Obligation de tourner à gauche	Obligation de tourner à droite	Impossible d'aller tout droit

Figure 16 – Formalisme employé pour les code-barres

Un code "doublé" serait composé de deux lignes elles mêmes séparées d'une épaisseur de ligne. Les codes seraient placés à 30cm du premier croisement.

Dès la lecture d'un code au sol, le robot sera donc "attentif" à l'éventuelle apparition d'un doublon pendant environ deux centimètres. Si rien n'est lu passé cette distance, il en déduira qu'il s'agit d'un code simple. Il prendra alors une décision de direction (et pourra au passage en avertir les autres robots) et commencera un décompte de distance afin de gérer les activations et désactivations de ses capteurs jusqu'au franchissement de l'intersection.

Illustrons ce système avec le cas de la bifurcation à gauche :

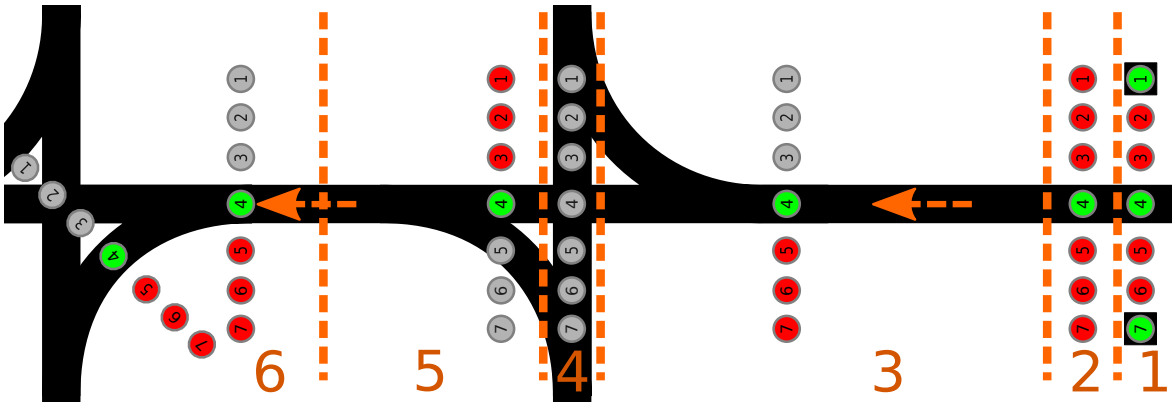


Figure 17 – Cas du virage à gauche sur un carrefour avec sept capteurs

1. Lecture d'un code-barre : dans 30cm le robot aura la possibilité de tourner à droite et à gauche.
2. Aucun doublon détecté dans les quelques centimètres suivants. La possibilité d'aller tout droit nous est donc offerte. Prise de décision quant à la direction (nous choisissons d'aller à gauche) et avertissement des autres robots en conséquence.
3. Le premier croisement ne nous intéresse pas. Nous allons continuer tout droit et éteignons donc la moitié droite de nos capteurs.
4. Au passage du croisement, nous éteignons temporairement l'ensemble de nos capteurs.
5. Le croisement passé, nous éteignons cette fois la moitié gauche de nos capteurs pour ne pas être perturbés par "la voie d'insertion".
6. Peu de temps après, nous rallumons la moitié gauche et éteignons la moitié droite des capteurs pour suivre la voie bifurquant à gauche.

Le robot doit donc être conscient des distances parcourues avec une précision supérieure à deux centimètres. Le moyen le plus simple d'implémenter cette fonctionnalité est sans aucun doute au travers d'un encodeur incrémental placé sur l'axe de propulsion du robot.

Nous pouvons donc considérer qu'à ce stade, nous disposons d'un solution satisfaisante répondant à la problématique du suivi de trajectoire. Cette solution répond précisément aux contraintes du projet, tout en conservant un certain niveau de simplicité. Elle est, pour rappel, constituée des éléments suivants :

- **Sur le circuit :** Un réseau de lignes blanches sur fond noir représentant les trajectoires empruntables par le robot.
- **Sur le robot :** Un ensemble de 7 réflecteurs optiques.

4.2.2 Détection d'obstacles et de de signalisation

4.2.3 Implémentation matérielle

4.2.3.1 Propulsion et direction La solution la plus répandue en "petite robotique" consiste en l'utilisation de deux moteurs à courant continu pilotés indépendamment assurant à la fois la propulsion et la direction : on réduira les gaz à droite pour tourner de ce côté et réciproquement. Ces moteurs sont généralement pilotés en "PWM" , mais d'autres solutions sont possibles et dépendront de notre contrôleur. Beaucoup d'applications utilisent des chenilles ou des courroies en caoutchouc pour assurer la propulsion, mais nous privilégierons l'utilisation de roues à pneus, par soucis de simplification de la conception (l'emploi de chenilles ou de courroies induit une grande précision d'usinage et de montage pour assurer le parallélisme des poulies et la tension de la courroie) et de réduction des coûts.

L'utilisation d'un seul moteur de propulsion et d'une roue directionnelle pilotée par un servomoteur a longtemps été considérée, mais cette solution s'est révélée bien plus complexe à mettre en place et sans réel intérêt particulier.

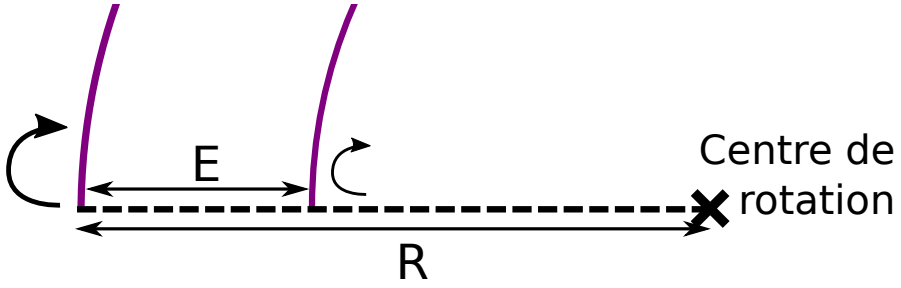


Figure 18 – Virage à l'aide de deux moteurs indépendants

L'espacement entre les deux roues E est constant et connu. On fixera la vitesse appliquée à la roue à l'extérieur du virage ω_{ext} comme référence. On adaptera donc la vitesse de la roue à l'intérieur du virage ω_{int} en fonction du rayon de virage R désiré. Il s'agit là d'une simple relation de Thales et on définit ainsi :

$$\omega_{int} = \omega_{ext} \frac{R - E}{R}$$

On peut donc obtenir n'importe quel rayon de virage en diminuant simplement la vitesse de la roue à l'intérieur du virage. L'approximation selon laquelle la tension appliquée à un moteur et sa vitesse de rotation sont linéairement liés sera amplement suffisante dans le cadre de ce projet.

4.2.3.2 Controleur Nous cherchons un contrôleur capable d'embarquer des applications en C (ou ses dérivés) et si possible en Python et/ou Matlab. Ce contrôleur devra être capable d'acquérir les données de différents capteurs et de commander deux moteurs au minimum (sans compter d'éventuelles sorties type LEDs). Le contrôleur devra également bénéficier d'une puissance de calcul suffisante pour les

opérations de reconnaissance d’image.

Ce dernier critère élimine d’office les microcontrôleurs 8 et 16 bits et donc les cartes de type "Arduino" dont la puissance de calcul et la capacité de traitement de flux est bien trop faible pour permettre une reconnaissance d’image supérieure à quelques pixels carrés.

Nous nous sommes naturellement tournés vers la famille des micro-ordinateurs mono cartes. Cette récente catégorie d’ordinateurs propose sur une carte de quelques centimètres de côté et pour moins d’une centaine d’euros un véritable ordinateur avec une connectivité réseau et USB, bien souvent vidéo, une puissance de calcul extrêmement confortable et la capacité de faire tourner des systèmes Linux et dérivés (et donc d’exécuter tous types de codes). Démocratisés en 2011 avec le Raspberry-Pi, ces ordinateurs se sont multipliés et l’offre est aujourd’hui très large. Nous avons ciblé notre sélection sur les plus repandus (qui sont, de par leur succès, bien plus abordables et faciles à se procurer, et bénéficient d’une plus large et solide base documentaire) : les Raspberry-Pis, Beaglebones et UDOOs (pour ne citer qu’eux).

Les UDOOs furent vite écartés en raison de leur prix. Le Raspberry-Pi 2 et BeagleBone Green se situent dans la même gamme de prix (autour de 40$). Nous avons arrêté notre choix sur le BeagleBone Green (BBG) : ce dernier, contrairement au Raspberry-Pi 2, ne dispose pas de sortie vidéo (dont nous n’avons pas utilité au sein de ce projet) et possède une puissance de calcul légèrement inférieur à ce dernier (tout en restant parfaitement respectable) mais possède une mémoire intégrée (ce qui dispense de l’achat d’une carte séparée) et surtout de capacités d’entrées/sorties numériques et analogiques bien supérieures. Ce point nous est extrêmement important : en effet, le Raspberry-Pi n’est capable de générer qu’un signal PWM et ne possède aucune entrée analogique ce qui aurait nécessité l’utilisation de "périphériques" supplémentaires pour la gestion des moteurs et des capteur et donc alourdi les coûts et la complexité de conception, augmenté le nombre de sources potentielles de pannes...

Nous utiliserons donc un BeagleBone-Green Wireless :

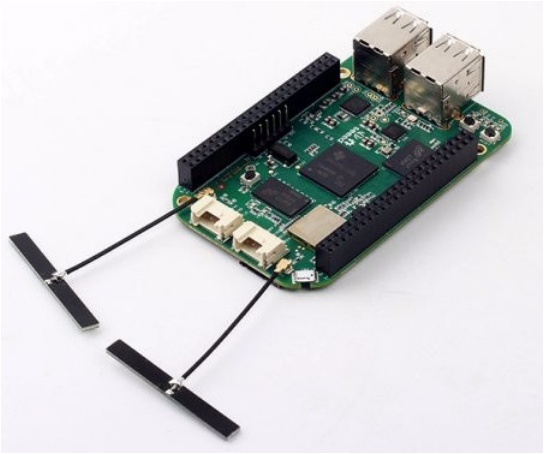


Figure 19 – Le BeagleBone Green ***(source : seeedstudio.com)***

Ce dernier est équipé d'un processeur ARM 32bits à 1Ghz, de 512Mo de RAM de 4Go de mémoire embarquée et d'un impressionnante capacité de communication : 7 entrées analogiques (la carte comprend un convertisseur analogique-numérique 12bits), 65 entrées/sorties numériques (0 ou 3.3V) et bénéficie même de fonctionnalités telles qu'un compteur d'impulsion intégré au processeur (particulièrement utile pour l'interfaçage de notre encodeur). La connectivité wifi et bluetooth offre de belles possibilités quant aux interfaces utilisateur et la capacité de communiquer entre robots. La carte consomme moins de 1,5W et coûte 45$ [2]. Nous la "chapeauterons" d'une carte d'extension que nous réaliserons nous même pour faire l'interface entre les ports d'extension et nos différents capteurs et actionneurs (voir 5.2.1.3, page 27).

Notons que le BBG doit être alimenté en 5V, et travaille avec un niveau logique "CMOS" de 3.3V. Les entrées et sorties numériques devront donc être compatibles (niveau haut supérieur à 2.5V et inférieur ou égal à 3.3V et niveau bas inférieur à 1.3V).

4.2.3.3 Suivi de ligne Comme dit précédemment, nous utiliserons sept réflecteurs optiques infrarouges pour effectuer le suivi de ligne. Un réflecteur optique est composé d'une LED (ici infrarouge) et d'un phototransistor. Le phototransistor est comparable à un transistor classique, dont le courant de base serait remplacé par une intensité lumineuse, ou plus simplement encore comme une résistance variable en fonction de l'intensité lumineuse reçue.

Nous utiliserons des TCRT5000, très répandus.

Les capteurs nous fournissent donc une tension variable qu'il faudra pouvoir exploiter : en dessous d'une certaine valeur de cette tension, nous pourrons conclure en la présence d'une ligne blanche (forte réflectivité).

Deux solution s'offrent à nous : ou bien concevoir un montage basé sur un comparateur et une résistance variable (pour régler la valeur seuil) afin d'obtenir une

sortie booléenne (par exemple : 5V en présence d'une ligne blanche et 0V sinon) qui pourra très simplement être lue sur n'importe quelle entrée numérique du contrôleur, ou bien tirer partie de l'encodeur analogique-numérique intégré au contrôleur pour lire la valeur directement. Nous privilégierons cette méthode qui permet de limiter le nombre de composants, et donc le coût et les sources de pannes.

Le BeagleBone Green possède sept entrées analogiques (nous avons justement sept capteurs) capables de discerner $2^{12} = 4096$ niveaux différents linéairement répartis entre 0 et 1,8V[2]. Ainsi, l'application d'une tension de 0V à l'une de ce entrées nous permettra de lire la valeur "0" logiciellement, 2048 pour 0,9V, 4096 pour 1,8V etc... Nous prendrons garde à ne jamais dépasser cette valeur sous peine d'endommager le controleur. Le BBG fournit une tension de 1.8V spécifiquement pour cette application[2].

La datasheet du TCRT5000 [12] nous apprend que la LED émettrice possède une tension directe de 1.25V et n'accepte que jusqu'à 60mA. Afin de ne pas surcharger la sortie à 1.8V fournie par le BBG (qui ne fournit pas plus de 50mA), nous alimenterons les LEDs en 5V via une résistance de 120Ω, faisant ainsi traverser la LED par $\frac{5-1.25}{120} = 31mA$, ce qui est idéal (intensité IR suffisamment élevée sans risquer "d'éblouir" les autres capteurs ou de griller la LED).

On connectera le collecteur du phototransistor à la ligne 1.8V au travers d'une résistance de pull-up de 2.7kΩ, la valeur la plus élevée de résistance nous permettant d'assurer un courant de collecteur de 5mA (comme recommandé dans la datasheet). Nous choisissons la valeur la plus élevée afin de limiter la consommation en courant, mais surtout d'augmenter la sensibilité du capteur (il sera ainsi plus facile au phototransistor de "tirer" la tension vers le bas).

Nous avons donc le schéma de principe suivant :

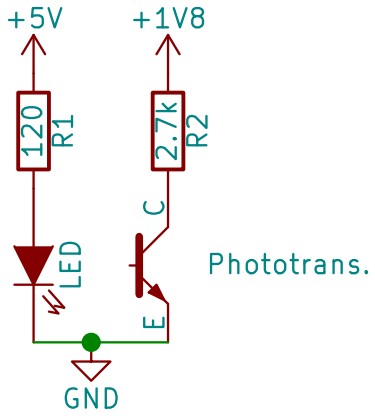


Figure 20 – Schéma de principe d'un réflecteur optique

L'intégration et l'agencement des capteurs sont discutés en 5.2.1.2 (page 27).

4.2.3.4 Acquisition d'images L'acquisition d'images, pour des raisons évidentes de coûts et de simplicité d'utilisation, sera réalisée à l'aide d'une webcam standard

de bureau, connectée en USB au BeagleBone. Nous avons donc choisi d'utiliser la Webcam Logitech C170 :

Figure 21 – Webcam Logitech C170 ***(source : Logitech)***

Compte tenu du fait que les caractéristiques techniques du BeagleBone ne nous permettait pas de traiter de la Haute Définition en quasi-temps réel, et que nous n'avions pas besoin d'un tel niveau de précision, nous nous sommes donc intéressés à cette caméra qui, en plus d'être d'une qualité acceptable, possède un clip de fixation universel, facilitant ainsi la mise en place sur le robot.

L'acquisition logicielle sera réalisée par un script réalisé en Python. Avec l'aide du package OpenCV, il est aisé d'interfacer la caméra ainsi que de paramétrer le flux vidéo (taille de l'image, nombre d'images par seconde...) afin d'alléger la charge de traitement de l'image pour ne pas déteriorer les performances du BBG. Les images ainsi récupérées (10/s) seront ensuite analysés selon plusieurs critères (couleurs et leur position, contours...) afin de participer à une prise de décision fiable et pertinente.

4.2.3.5 Évaluation des distances

Distances parcourues L'évaluation des distances parcourues se fera simplement au travers d'un encodeur incrémental placé sur l'un des deux axes de propulsion du robot. Un encodeur incrémental produit au moyen de capteurs optiques ou magnétiques un signal carré dont chaque période correspond au passage d'un élément de référence devant son capteur. Ainsi, si on place une roue comprenant quatre éléments de référence (4 aimants, ou 4 obturateurs) régulièrement espacés sur l'axe d'un moteur et le capteur judicieusement, nous pourrons observer un front montant chaque fois que le moteur effectue un quart de tour. En comptant à la fois les fronts montants et descendants, nous connaîtrons les variations angulaires du moteur avec une précision d'un huitième de tour. Et en plaçant un second capteur en quadrature (en déphasage de 90 degrés), notre précision sera doublée.

Le constructeur de nos motoréducteurs, Pololu, propose dans son catalogue un ensemble compatible avec notre motoréducteur comprenant un encodeur à deux capteurs optiques en quadrature et des roues intégrant une mire optique adaptée.

Figure 22 – Ensemble motoréducteur, roue et encodeur ***(source : Pololu)***

Cette solution nous garantie une compatibilité directe, et nous évite surtout d'avoir à adapter un encodeur (souvent coûteux) sur l'axe entre la roue et le réducteur. L'utilisation des deux capteurs en double fronts nous offre une résolution de 48 impulsions par tour de roue. La roue ayant un diamètre de 42mm, cela nous donne une précision de lecture de :

$$\frac{\pi \times 42}{48} = 2.75mm$$

Nous pourrons même nous permettre de n'utiliser qu'un capteur et bénéficier ainsi d'une résolution de 5.5mm, répondant largement à nos critères.

Il s'agira ensuite de compter le nombre de fronts montants et descendants sur le signal généré par le capteur. Ceci peut se faire simplement en connectant le capteur à une entrée numérique du contrôleur et en observant les changements d'état de cette entrée, mais cette méthode consomme énormément de ressources pour une tâche aussi simple. Une autre solution, quand le processeur le permet, consiste à demander au processeur de détecter les fronts électroniquement et de déclencher un script (compteur) le cas échéant. Cette solution est évidemment préférable lorsqu'elle est disponible.

Par chance, notre processeur ARM-A8 possède cette fonctionnalité et intègre même un module eQEP[11] justement dédié au traitement des signaux d'encodeurs en

quadrature et qui fera tout le travail à notre place (pour peu que l'on utilise un noyau Linux compatible avec le driver de ce module).

Distance des obstacles La distance avec les obstacles en avant sera mesurée grâce à un sonar HC-SR04. Ce capteur a l'avantage d'être facilement utilisable et est suffisament précis, compte-tenu de notre besoin. De plus, il est également peu sujet aux interférences.

Figure 23 – Sonar ultrasons HC-SR04 ***(source : Instructable)***

Cette fonction sera particulièrement utile afin de s'assurer que le robot ne rentrera pas en collision avec un potentiel robot situé sur sa trajectoire.

Le principe de fonctionnement est simple : on mesure le temps t_{echo} séparant une impulsion envoyée ("trigger") et le retour de cette impulsion ("echo"). Un simple calcul nous donne ensuite la distance à l'obstacle : $d_{obstacle} = \frac{t_{echo} \times v_{son}}{2}$, avec v_{son} la vitesse du son égale à $340 m.s^{-1}$, selon les Conditions Normales de Température et de Pression.

4.2.3.6 Signalisation de ses intentions Nos robots seraient considérés, en théorie de l'intelligence artificielle, comme des "agents". Ils en possèdent en effet toutes les caractéristiques et notamment celle de l'autonomie. Chaque robot est autonome dans le sens où il prend ses propres décisions sans recevoir de commande extérieure. Il est donc primordial que chaque robot puisse avertir les autres de ses intentions, et lui même décrypter les intentions des autres agents.

Nous pourrions atteindre cet objectif en faisant communiquer les robots via un réseau quelconque, mais cela sous-entendrait que ces derniers ne soient capables d'évoluer qu'au milieu de leurs "congénères", ce qui serait parfaitement irréaliste dans la vraie vie. Nous avons donc décidé d'implémenter de simples clignotants et feux stops à LED, simples à mettre en œuvre et universels.

4.2.3.7 Interaction avec l'utilisateur Ayant constaté au cours de travaux pratiques qu'il est souvent peu aisé de gérer le déclenchement des séquences de programmes (obligation de passer par des temporisateurs et de brancher/débrancher l'alimentation de la carte), nous avons décidé d'implanter un simple bouton poussoir (qui pourra servir pour démarrer, mettre en pause ou arrêter un programme, par exemple) ainsi qu'un interrupteur commandant l'alimentation de la carte mère (et donc de l'ensemble du système).

4.2.3.8 Batterie Pour rappel, voici la liste des composants qu'il nous faudra alimenter :

- Deux moteurs à courant continu
  D'après leur fiche technique[10], ces derniers fonctionnent entre 3 et 9V avec un rendement optimal autour de 6V. Leur consommation "axe bloqué" (en couple maximum) est de 1,6A. Une estimation pessimiste de leur consommation en fonctionnement normal consistant à prendre le quart de ce courant, nous pouvons considérer que chaque moteur consommera en moyenne moins de 400mA.
- Un BeagleBone Green
  Le BBG doit être alimenté en 5V et a une consommation moyenne de 400mA [5].
- Sept capteurs IR
  Chaque capteur consommant, en tout, environ 35mA.
- Une moyenne (généreuse) de 2 LEDs allumées
  Avec une consommation d'environ 20mA par LED.
- Divers petits composants
  Pour lesquels on fera un devis global pessimiste à 100mA.

Soit une consommation moyenne extrêmement pessimiste de 1600mA. La plus haute tension nécessaire (demandée par les moteurs) est d'environ 6V.

Étant donné qu'il est plus difficile de réduire une tension que de l'augmenter, et un moteur à courant continu étant beaucoup plus souple que le reste des circuits électroniques (le moteur ne souffrira pas de tourner à une tension légèrement inférieure) nous privilégierons l'usage d'une batterie de 6V ou plus et d'une capacité supérieure à 1600mAh.

4.2.4 Implémentation logicielle

Nous ne rentrerons ici pas dans les détails, d'une part car le but n'est pas de procéder à la partie développement dans ce dossier et d'autre part car la partie logicielle représente la part la plus fluctuante de nos robots, destinés à être reprogrammés tout au long de leur cycle de vie. Cependant, nous allons tenter de détailler rapidement l'approche que nous recommandons au sujet du développement logiciel dans le cadre de ce projet.

4.2.4.1 Suivi de ligne Le suivi de ligne, en terme de programmation, est une simple problématique de régulation. Nous recueillerons les données issues des différents capteurs, les traiterons, et produirons une réponse au travers des actionneurs.

La question, au delà de celle des logiques et algorithmes utilisés pour produire cette régulation, ici hors sujet, est celle de l'interaction avec les capteurs et actionneurs.

Nos programmes devront faire l'acquisition des données suivantes :

- Données issues des réflecteurs, sous la forme de la valeur de sept tensions.
- Données issues de l'encodeur, sous la forme d'un comptage des pulsations.

Ils devront communiquer leurs ordres en faisant varier la tension aux bornes des moteurs. Ceci se fera au travers de la création d'un signal "PWM".

Il s'agira donc d'établir le lien entre les interfaces d'entrées/sorties du contrôleur et le code à proprement parler.

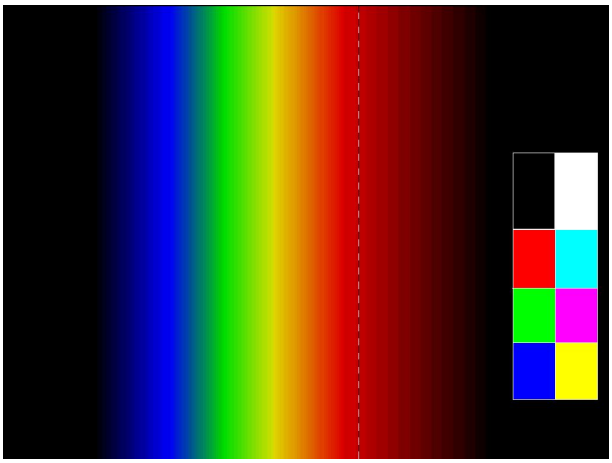
Sur le BBG, l'un des moyens les plus simples d'effectuer cela est de passer par la bibliothèque Python crée à cet effet par Justin Cooper [3] [4]. Cette bibliothèque simplifie grandement ce genre d'opérations, et lire la valeur (numérique ou analogique) d'un port, ou changer cette valeur ne prend que deux lignes de code (se référer à la documentation).

Une utilisation des entrées/sorties reste néanmoins possible et accessible en C et C++, par exemple [6], mais nous favoriserons l'utilisation de Python pour des raisons "d'affinité", mais également car ce langage simplifie l'utilisation de nombreux outils nécessaires à ce projet, notamment la communication réseau et le traitement d'images, et ne nécessite aucune compilation.

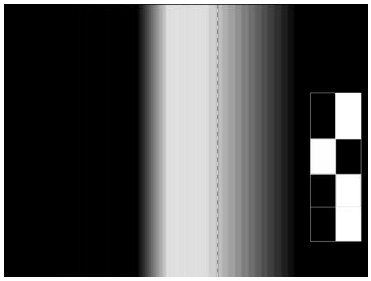
Nous veillerons cependant, toujours dans un soucis de modularité, à ce que nos "modules" python, soient compatibles avec d'éventuelles réalisations en C, java ou autre (voir 5.2.2 : Intégration Logicielle, page 27).

4.2.4.2 Reconnaissance d'image La reconnaissance d'image sera utilisée à plusieurs fins :

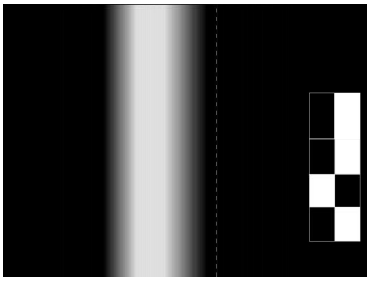
- Couleur du feu La détection de la couleur du feu est un aspect primordial de ce projet. Grâce au package OpenCV, il est très facile de décomposer le spectre de la lumière visible, et donc de déterminer s'il y a en face de nous une lumière verte ou rouge, comme le montre la figure suivante : http ://physique-eea.ujf-grenoble.fr/



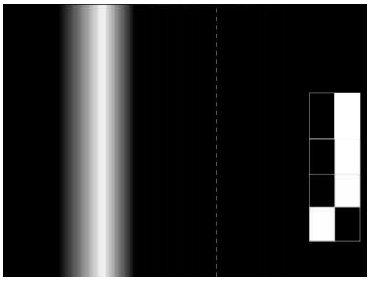
(a) Spectre de la lumière visible



(b) Filtre rouge



(c) Filtre vert



(d) Filtre bleu

Figure 24 – Spectre (a), selon différents filtres (b),(c) et (d).
(source : http ://physique-eea.ujf-grenoble.fr/)

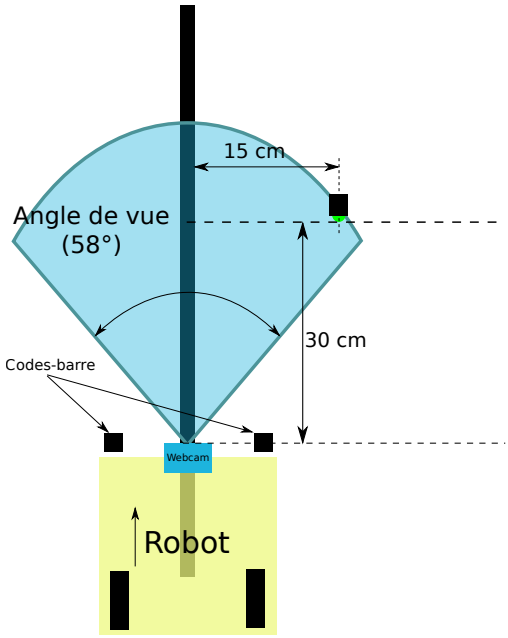


Figure 25 – Schéma de principe de l'approche d'un feu

Connaissant l'angle de vue de la caméra (58°), et en utilisant des fonctions géométriques simples, nous pouvons déterminer rapidement la distance minimale au feu s'il faut s'arrêter : $d_{min} = 15cm \times tan(\frac{58}{2}) = 8,3cm$

- Présence et intentions des autres utilisateurs

5 Intégration

5.1 Circuit

5.2 Robot

5.2.1 Intégration Matérielle

5.2.1.1 Ensembles propulsifs *[Motoréducteur + roue + encodeur (parler conversion tensions)]*

5.2.1.2 Carte Réflecteurs Optiques *[Espacement des capteurs + hauteur sur piste]*

Nous connecterons la carte au moyen d'une nappe HE10 à 10 connecteurs.

Le schéma électronique et les gerbers du circuit imprimés sont disponibles en annexe A (page 30)

5.2.1.3 Carte-Mère **[MAXIME]**

5.2.1.4 Structure du robot *[Profilés alu standards, vis idem...]*

5.2.1.5 Intégration globale, maquette numérique *[Vues Catia et breve explication]*

5.2.2 Intégration Logicielle

Modules indépendants à 4 niveaux (acquisition, traitement, décision, action), les sorties des uns servant d'entrées aux autres (mux) communicant via socket UDP => possibilité de faire interagir des programmes en C, en Python, en Java...
+ un programme "lanceur et ordonnanceur" avec son fichier de config. Interface web pour programmation et paramétrage.
Modules "de base" livrés sous forme de classes avec nombreuses méthodes fournies.
IMPORTANCE DE LA DOCUMENTATION

6 Estimation du coût de mise en place et faisabilité

7 Conclusion et aperçu des évolutions possibles

Annexes

A Carte Réflecteurs Optiques

A.1 Schéma électronique

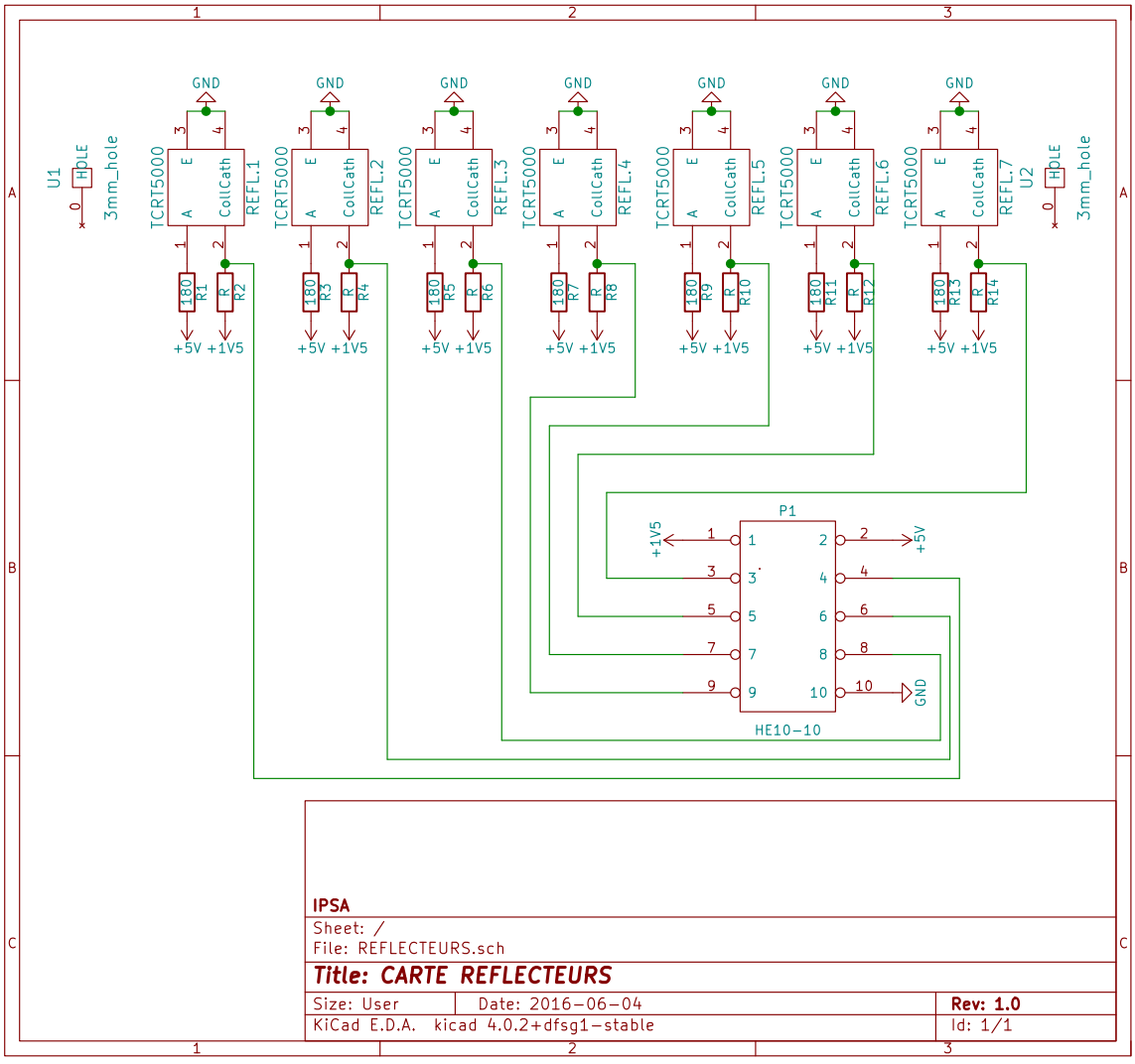
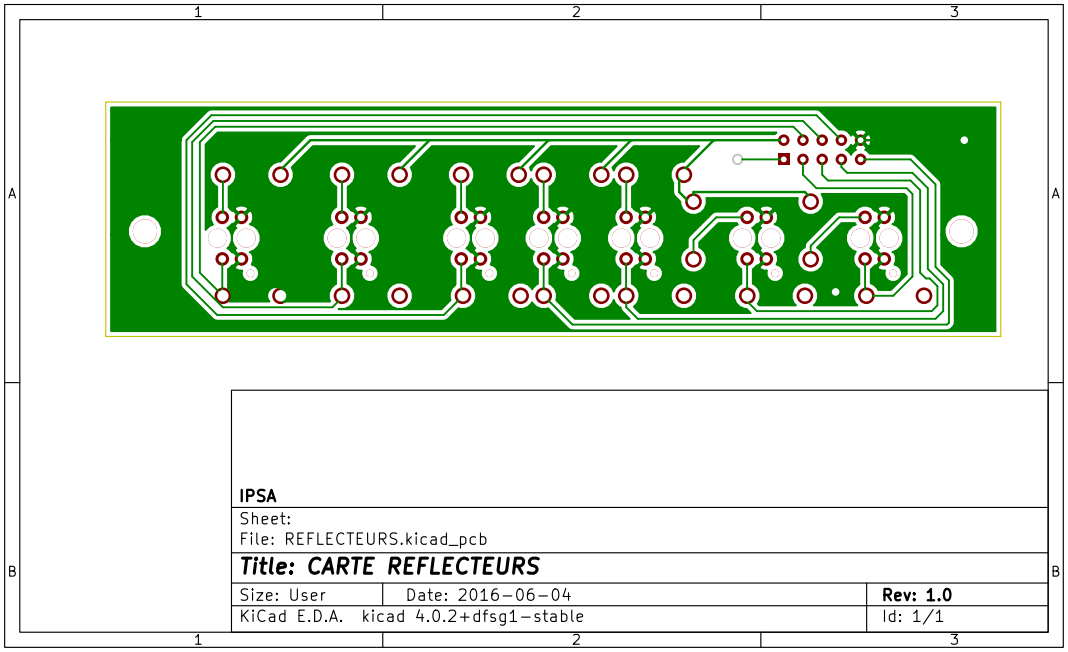


Figure 26 – Schéma électronique de la carte Réflecteurs Optiques

A.2 Gerbers



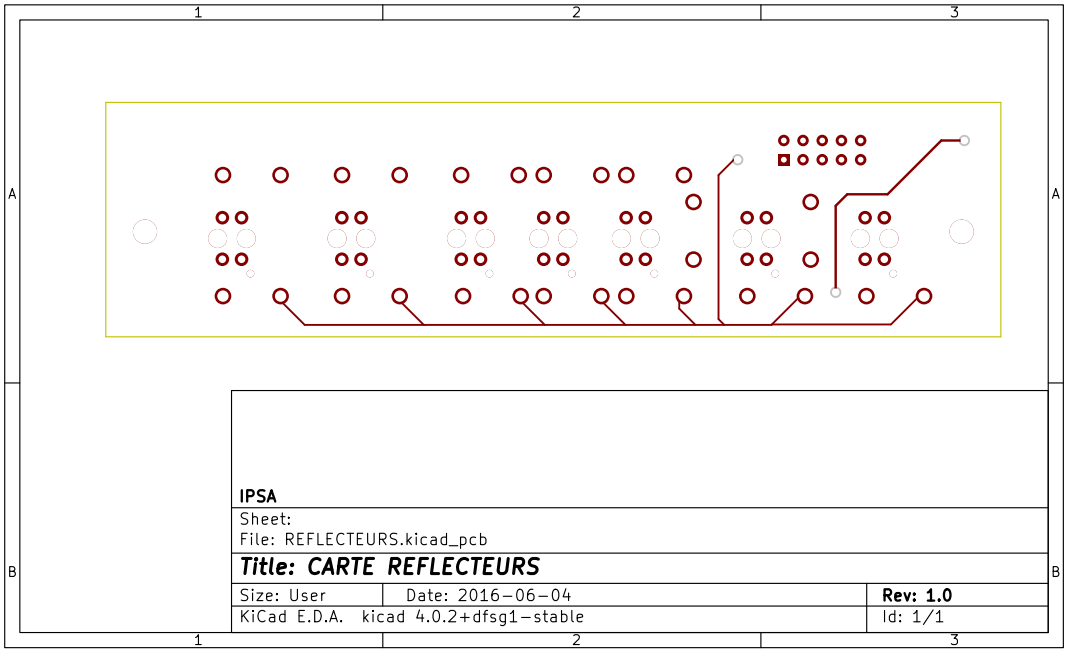


Figure 27 – Gerbers de la carte Réflecteurs Optiques

Références

Nomenclature

BBG BeagleBone Green, page 18

eQEP Enhanced Quadrature Encoder Pulse, ou Pulsation d’Encodeur en Quadrature Améliorée, page 22

IR Infra-Rouge, page 20

LED Light-Emitting Diode, ou Diode Electro-Luminescente, page 19

PID Proportionnel, Intégrateur, Dérivateur, page 11

PWM Pulse-Width Modulation, ou Modulation de Largeur d’Impulsions, page 17

TP Travaux Pratiques, page 3

Bibliographie

[1] Line tracking sensors and algorithms. https://www.ikalogic.com/line-tracking-sensors-and-algorithms/, 2008.

[2] Beaglebone green wireless wiki. http://www.seeedstudio.com/wiki/Beaglebone_green_wireless, 2016.

[3] J. COOPER. Setting up io python library on beaglebone black. https://learn.adafruit.com/setting-up-io-python-library-on-beaglebone-black/gpio, 2015.

[4] J. COOPER. Adafruit beaglebone io python - readme. https://github.com/adafruit/adafruit-beaglebone-io-python, 2016.

[5] T. DICOLA. Embedded linux board comparison - power usage. https://learn.adafruit.com/embedded-linux-board-comparison/power-usage, 2015.

[6] D. MOLLOY. Chapter 6 : Interfacing to the beaglebone input/outputs chapter 6 : Interfacing to the beaglebone input/outputs. http://exploringbeaglebone.com/chapter6/#The_BealeBone_GPIOs_8211_Using_Device_Tree_Overlays, 2015.

[7] M. BENKIRANE N. EL JAYIDI. *µRobot : Le robot suiveur de ligne*. PhD thesis.

[8] Eric Nunès. Bientôt, des voitures "intelligentes". *Le Monde*, 2003. http://www.lemonde.fr/societe/article/2003/10/28/bientot-des-voitures-intelligentes_339891_3224.html.

[9] S. MONK P. SCHERZ. *Practical Electronics for Inventors*. McGraw-Hill Education TAB, 2016.

[10] Pololu. *HP 294*, 2016. https://www.pololu.com/product/994.

[11] Texas Instrument. *eQEP*, 2006. http://www.ti.com/lit/an/spraah1/spraah1.pdf.

[12] Vishay Semiconductors. *TCRT5000*, 2009. http://www.vishay.com/docs/83760/tcrt5000.pdf.

Table des illustrations

Résumé