Alexandre LY – 71701881
Julien CHAU – 22018303

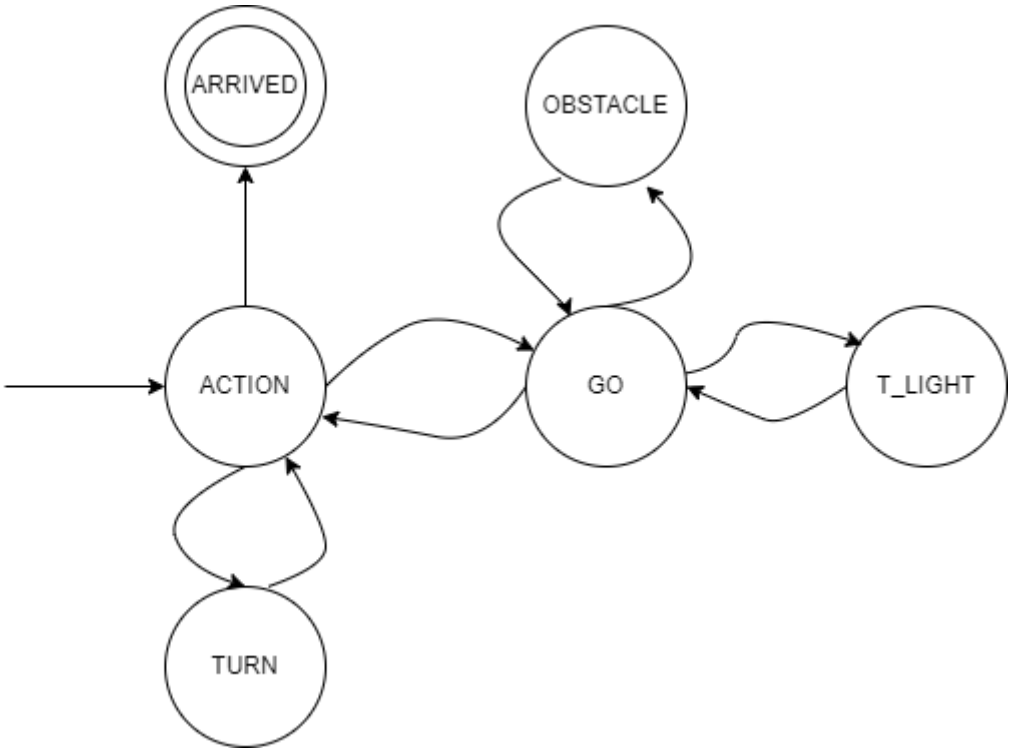
RAPPORT - Projet de Programmation Synchrone

1. Introduction

Le but du projet est de programmer le module de contrôle d'un véhicule autonome qu'on place sur la cartographie d'une ville virtuelle en deux dimensions. Cette carte est composée d'un circuit tracé en couleur, et ce parcours doit respecter certaines contraintes : une limitation de vitesse, des feux rouges, des obstacles à éviter, et des changements de directions.

Le programme sera écrit en Heptagon et devra suivre la méthodologie des langages synchrones vu en cours.

2. Architecture du contrôleur



Le contrôleur est constitué d'un automate à états : *Action*, *Arrived*, *Go*, *Turn*, et *Obstacle*. A chaque passage de bande verte du parcours par la voiture, on entre dans l'état initial *Action* de l'automate : selon le type de l'action, on entre dans l'état correspondant (*Go, Turn, Arrived*), après chaque action exécutée, on incrémente la variable globale de l'index du tableau des actions. Tandis que lorsque la voiture capte une bande rouge, cela

correspond à un feu de signalisation et la voiture doit s'arrêter : on entre dans l'état *TrafficLight*. Le sonar de la voiture permet de déterminer s'il existe un obstacle proche de la voiture, si oui alors on entre dans l'état *Obstacle* et on doit s'arrêter pour éviter une colision.

3. **Algorithme de suivi de ligne**

Pour que la voiture puisse suivre le parcours à travers la ville, sans faire de hors-piste, on se retrouve face à un problème : lorsque la voiture se retrouve d'un côté de la route, comment faire pour retrouver le bon sens ? Une méthode naïve serait que la voiture tourne dans le sens opposé pour revenir sur le bon chemin, avec une valeur discrète. Mais cette technique n'est pas efficace puisqu'on se retrouve avec une voiture qui roule constamment en « zig-zag ».

L'algorithme du régulateur PID (Proportionnal-Integral-Derivative) est un algorithme efficace pour la résolution de ce problème.

Plus la voiture dévie de son chemin, et plus les capteurs perçoivent la couleur du côté vers lequel elle se dirige : cyan pour la gauche et magenta pour la droite. Cette différence de couleur avec le bleu, couleur de la route, est l'erreur ; et plus cette variable est grande, plus la correction est élevée.

L'algorithme du PID corrige cette valeur de l'erreur à l'aide d'une correction basé sur la proportionnalité, l'intégral et la dérivée :

1. P : plus le taux d'erreur est large, et plus la correction est importante, selon un facteur de gain.
2. I : après correction via P, s'il existe toujours une erreur résiduelle, alors l'intégrale permet de corriger selon les valeurs d'erreurs dans le temps.
3. D : enfin, la dérivée est une estimation de la valeur future de l'erreur, basé sur sa fréquence de changement.

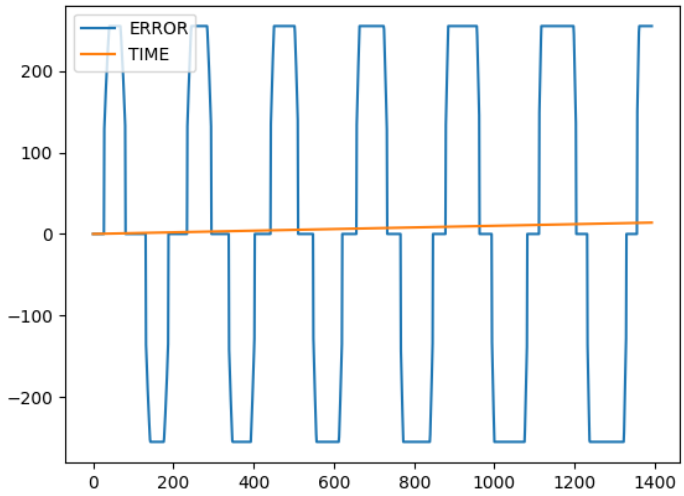
L'algorithme utilisé est une méthode alternative de réglage du modèle de régulateur PID, il s'agit de la méthode Ziegler-Nichols.

La formule de l'implémentation du PID paramétré avec la méthode Ziegler-Nichols :

$$PID = 0.60Kc + \frac{2K_p dT}{P_c} + \frac{K_p P_c}{(8dT)}$$

K_c : $gain\ maximal$
P_c : $période\ d'oscillation$

Alexandre LY – 71701881
Julien CHAU – 22018303



Graphe des oscillations des valeurs de l'erreur dans le temps.

4. Turn

Pour les actions *Turn*, pour calculer le temps nécessaire à la voiture pour tourner d'un certain degré à une certaine vitesse, on utilise les formules suivantes :

a. $Circumference = \pi * d$
b. $d_{arcLength} = \left(\frac{degree}{360}\right) * Circumference$
c. $t = \frac{d}{v}$

- On calcule tout d'abord la circonférence de la voiture en multipliant le diamètre (distance entre les deux roues) avec π.
- Ensuite, on calcule la longueur de l'arc qu'on obtiendra en faisant un tour d'une certaine valeur de degré avec la deuxième formule.
- Enfin, on calcule le temps nécessaire en divisant la longueur de l'arc par la vitesse de la voiture.

Alexandre LY – 71701881
Julien CHAU – 22018303

5. Remarques

- Le capteur ventral détecte la couleur des marquages des actions et des feux de signalisation, différente de celle de la route, ce qui peut fausser les calculs de corrections d'erreurs du PID. La solution utilisée est de garder et réutiliser avec le mot clé *last* les valeurs calculées auparavant.

```
last error : float = 0.0;
last integral : float = 0.0;
last derivative : float = 0.0;
```