



ÉCOLE POLYTECHNIQUE  
FÉDÉRALE DE LAUSANNE

# Simulation du robot Thymio II

Rapport de Projet semestriel (Automne 2013)



**DINH Nicolas**

January 10, 2014

Professeur responsable:  
Francesco Mondada

Assistants responsables:  
Fanny Riedo, Stéphane Magnenat

# Contents

<b>1 Introduction</b>	<b>6</b>
1.1 Qu'est ce que le Thymio II ? . . . . .	6
1.2 Interface . . . . .	7
1.3 Objectifs du projet : Donnée et travail demandé . . . . .	10
1.4 Description du robot . . . . .	10
1.5 Délimitation des objectifs du projet . . . . .	12
1.6 Matériel de test . . . . .	12
<b>2 Modélisation des capteurs horizontaux</b>	<b>13</b>
2.1 Caractéristique des capteurs . . . . .	13
2.1.1 Mesures et observations . . . . .	13
2.1.2 Modélisation et choix du modèle . . . . .	14
2.2 Caractéristique du bruit des capteurs . . . . .	16
2.2.1 Mesures et observations . . . . .	16
2.2.2 Modélisation et choix du modèle . . . . .	16
<b>3 Modélisation des moteurs DC</b>	<b>18</b>
3.1 Vitesse des moteurs . . . . .	18
3.1.1 Détermination de la vitesse maximale des moteurs . . . . .	18
3.1.2 Caractéristique du profil de vitesse des moteurs . . . . .	19
3.1.3 Caractéristique du bruit des moteurs . . . . .	21
3.2 Le PWM . . . . .	23
3.2.1 Caractéristique du PWM . . . . .	23
3.2.2 Caractéristique du bruit du PWM . . . . .	25
3.3 Le courant . . . . .	27
3.3.1 Courant maximum $i_{max}$ . . . . .	27
3.3.2 Caractéristique du courant i . . . . .	29
3.3.3 Caractéristique du bruit du courant i . . . . .	30
3.4 Proposition d'amélioration pour les moteurs . . . . .	31
<b>4 Modélisation des capteurs verticaux</b>	<b>32</b>
4.1 Caractéristique des capteurs . . . . .	32
4.1.1 Mesures et observations . . . . .	32
4.1.2 Modélisation et choix du modèle . . . . .	36
4.2 Caractéristique du bruit des capteurs . . . . .	37
4.2.1 Mesures et observations . . . . .	37
4.2.2 Modélisation et choix du modèle . . . . .	37
<b>5 Modélisation des LEDs</b>	<b>38</b>
5.1 Mesures et observations . . . . .	38
5.2 Modélisation et choix du modèle . . . . .	39
<b>6 Implémentation du code dans enki et Aseba Playground</b>	<b>41</b>
6.1 Première implémentation d'un Thymio dans Enki et Aseba . . . . .	41
6.1.1 Lien entre Aseba et Enki pour le Thymio II . . . . .	41
6.1.2 Définition de la forme physique du robot . . . . .	41
6.1.3 Design 3D sous Blender . . . . .	42
6.2 Les événements . . . . .	43
6.3 Les capteurs horizontaux . . . . .	44
6.4 Les moteurs . . . . .	46

6.5	Les capteurs au sol . . . . .	51
6.6	Les LEDs . . . . .	55
<b>7</b>	<b>Conclusion</b>	<b>60</b>
<b>8</b>	<b>Annexes</b>	<b>I</b>
8.1	Annexe A : Code Matlab pour déterminer les paramètres optimaux pour les capteurs IR . . . . .	I
8.2	Annexe B : Résultats du code Matlab pour les paramètres des capteurs IR $x_0$ , $m$ et $c$ . . . . .	II
8.3	Annexe C : Tableau de calcul du retard $\tau$ (en seconde) pour le profil de vitesse des moteurs . . . . .	II
8.4	Annexe D : Tableau de calcul du retard $\tau$ (en seconde) pour le profil de courant des moteurs . . . . .	III
8.5	Annexe E : Liste des fichiers modifiés ou ajoutés dans les bibliothèques Enki et Aseba . . . . .	III
8.6	Annexe F : Définitions des nouvelles classes . . . . .	IV
8.7	Annexe G : Mesures complémentaires pour les capteurs horizontaux . . . . .	XI
8.8	Annexe H : Mesures complémentaires pour les capteurs verticaux . . . . .	XII

## List of Tables

1	Portée en cm des 7 capteurs pour chaque robot . . . . .	15
2	Bruit (ou écart type) exprimé en ua (unité aseba) des capteurs n°2 des 3 robots	17
3	Portée en cm des 7 capteurs pour chaque robot . . . . .	19

## List of Figures

1	Le Thymio II est un jouet et un outil éducatif (Source : Wikidot Thymio) . . . . .	6
2	Exemple de construction mécanique sur le Thymio II (Source : Wikidot Thymio)	6
3	Interface du logiciel Aseba ainsi que divers robots programmables (Source : Wikidot Thymio) . . . . .	7
4	Interface détaillée du logiciel Aseba Studio . . . . .	8
5	Fenêtre de connexion avec un robot . . . . .	9
6	Emplacement des différents capteurs et actuateurs (Source : Wikidot Thymio)	11
7	Emplacement des 7 capteurs horizontaux sur le robot Thymio II . . . . .	13
8	Mesure de la valeur des capteurs (en unité aseba) du robot Thymio II TH095 en fonction de la distance (en cm) . . . . .	13
9	Caractéristique de la valeur des capteurs (en unité aseba) en fonction de la distance (en cm) . . . . .	15
10	Mesure du bruit des capteurs(en unité aseba) en fonction de la distance (en cm) . . . . .	16
11	Mesure de la vitesse maximale des moteurs (en ua) du TH085 en fonction du temps (en sec) pour une vitesse de consigne de -600ua . . . . .	18
12	Profil de la vitesse (en unité aseba) en fonction du temps (en seconde) . . . . .	19
13	Modèle de profil de vitesse proposé pour la simulation (en unité aseba) en fonction du temps (en seconde) . . . . .	20

14	Bruit des moteurs (écart type des données) exprimé en ua en fonction de la vitesse imposée (en unité aseba) . . . . .	21
15	Bruit des moteurs (écart type des données) exprimé en ua avec leur droite caractéristique en fonction de la vitesse imposée (en unité aseba) . . . . .	22
16	Modèle linéaire du bruit exprimé en ua en fonction de la vitesse imposée (en unité aseba) . . . . .	22
17	Valeurs du PWM (en ua) en fonction de la vitesse (en ua) . . . . .	23
18	Valeurs du PWM en moyenne pour (en ua) en fonction de la vitesse (en ua)	24
19	Valeurs du PWM en moyenne pour (en ua) en fonction de la vitesse (en ua)	24
20	Bruit du PWM (écart type) exprimé en ua en fonction de la vitesse (en ua) .	25
21	Comparaison entre le modèle et les mesures du bruit du PWM (écart type) exprimé en ua en fonction de la vitesse (en ua) . . . . .	26
22	Mesure du courant (en unité aseba) en fonction du temps (en seconde) pour des vitesses de consignes différentes . . . . .	27
23	Mesure du courant moyen (en unité aseba) en fonction de la vitesse (en unité aseba) . . . . .	27
24	Comparaison entre le modèle et le courant mesuré (en unité aseba) en fonction de la vitesse (en unité aseba) . . . . .	28
25	Mesure du courant (en unité aseba) en fonction du temps (en sec) . . . . .	29
26	Mesure du courant (en unité aseba) en fonction du temps (en sec) . . . . .	29
27	Mesure du bruit du courant (en ua) en fonction de la vitesse (en ua) . . . . .	30
28	Comparaison entre le modèle et les mesures sur le bruit du courant (en ua) en fonction de la vitesse (en ua) . . . . .	31
29	Page 1 des niveaux de gris entre 100% et 30% . . . . .	32
30	Différence du ressorti des 4 premiers niveaux de gris (entre 100% et 70%) selon les imprimantes respectivement citées précédemment . . . . .	33
31	Différence entre les valeurs mesurées par les capteurs en unité aseba en fonction du niveau de gris théorique (entre 0 et 1) pour les 4 feuilles avec le robot TH091 : 1. imprimante noir et blanc, 2. imprimante couleur EPFL, 3. imprimante couleur "maison", 4. imprimante couleur du labo LSRO . . . . .	33
32	Mesure des 3 variables lumière ambiante, lumière réfléchie et le delta (entre la lumière réfléchie et la lumière ambiante) exprimés en unité aseba en fonction du niveau de gris théorique . . . . .	34
33	Mesure dans l'obscurité des 3 variables des 2 capteurs (lumière ambiante, lumière réfléchie et la différence des deux exprimés en unité aseba) en fonction du niveau de gris théorique pour le robot TH085 (1), TH091 (2) et TH095 (3) .	35
34	Modèle du niveau de gris (en unité aseba) en fonction du niveau de gris théorique .	36
35	Bruit des capteurs au sol (en unité aseba) en fonction du niveau de gris théorique dans l'obscurité. Robots : TH085 (1), TH091 (2) et TH095 (3) .	37
36	Robot Thymio II dont 2 LEDs des boutons sont allumés avec différentes intensités . . . . .	38
37	Première tentative d'affichage des textures des LEDs . . . . .	39
38	Affichage des LEDs selon le code défini précédemment . . . . .	39
39	Position des points définissants la forme physique du Thymio II par rapport au centre du support crayon . . . . .	41
40	Design simplifié du corps du Thymio II sous Blender . . . . .	43
41	Affichage du Thymio II sous Aseba Playground . . . . .	43
42	Comparaison entre la réponse d'un robot réel et d'un robot simulé . . . . .	46
43	Profil de courant arbitraire en fonction du temps . . . . .	47

44	Profil de vitesse, du PWM et du courant pour le robot réel en fonction du temps . . . . .	50
45	Profil de vitesse, du PWM et du courant pour le robot simulé en fonction du temps . . . . .	50
46	Condition de mesure pour le robot TH095 . . . . .	53
47	Condition de mesure dans l'univers simulé . . . . .	53
48	Résultat de mesure des capteurs au sol du Thymio TH095 . . . . .	54
49	Résultat de mesure des capteurs au sol du modèle simulé . . . . .	54
50	Comparaison de l'affichage des LEDs, vue sur l'avant et le dessus des robots	58
51	Comparaison de l'affichage des LEDs, vue sur l'avant et le dessus des robots	58
52	Comparaison de l'affichage des LEDs, vue sur l'avant et le dessus des robots	58
53	Comparaison de l'affichage des LEDs avant des capteurs sur le robot simulé et le Thymio TH095 . . . . .	59
54	Mesure de la valeur des capteurs (en unité aseba) du robot Thymio II TH085 en fonction de la distance (en cm) . . . . .	XI
55	Mesure de la valeur des capteurs (en unité aseba) du robot Thymio II TH091 en fonction de la distance (en cm) . . . . .	XI
56	Différence entre les valeurs mesurées par les capteurs en unité aseba en fonction du niveau de gris théorique (entre 0 et 1) pour les 4 impressions avec le robot TH085 . . . . .	XII
57	Différence entre les valeurs mesurées par les capteurs en unité aseba en fonction du niveau de gris théorique (entre 0 et 1) pour les 4 impressions avec le robot TH085 . . . . .	XII

# 1 Introduction

## 1.1 Qu'est ce que le Thymio II ?

Le Thymio II est un robot programmable à but éducatif et accessible au grand public. Le robot est équipé de différents capteurs, de deux moteurs, de boutons et de témoins lumineux. Sa taille est de  $110 \times 110 \times 50\text{mm}$  et il est vendu entre 120chf et 135chf (selon le pays de livraison). Il est l'évolution du robot Thymio et ils ont tous les deux été développés dans le cadre de la collaboration entre le groupe MOBOTS (Miniature Mobile Robots Group) du laboratoire LSRO (Laboratoire de Systèmes Robotiques) de l'Ecole Polytechnique Fédérale de Lausanne (EPFL) et l'Ecole Cantonale d'Art de Lausanne(écal).



Figure 1: Le Thymio II est un jouet et un outil éducatif (Source : Wikidot Thymio)

Le code du Thymio est distribué en open source (c'est-à-dire de libre accès) et son matériel est produit et distribué à faible coût.

Son but étant l'éducation de la robotique au grand public, il est facile à programmer à l'aide de l'interface "Aseba" (décrite dans la partie 1.2) à travers deux types de programmation : la programmation graphique pour les débutants ou la programmation textuelle pour les initiés. Il permet aussi l'apprentissage de certains concepts technologique ou mécanique, et il est aussi possible d'y ajouter des constructions personnelles à l'aide de multiples fixations mécaniques (voir fig. 2).



Figure 2: Exemple de construction mécanique sur le Thymio II (Source : Wikidot Thymio)

## 1.2 Interface

L’interface de programmation du robot se fait à travers l’interface *Aseba Studio* qui est un logiciel open-source et destiné à la recherche et à l’enseignement de la robotique. Aseba est le lien entre le robot et l’utilisateur : il permet aux débutants de programmer des robots réels ou simulés facilement et efficacement à l’aide d’un langage de programmation très accessible.

Il y est aussi possible de contrôler les robots en temps réel en modifiant leurs variables de contrôle (par exemple l'attribution d'une consigne de vitesse pour la roue gauche).

Les robots simulés peuvent être visionnés à travers divers interfaces d'aseba tels que Aseba Playground ou Aseba Challenge (fenêtre de droite sur la figure 3). En particulier, l'outil Aseba Playground permet simplement d'observer des robots dans un univers prédéfini (composé d'obstacles et de texture au sol).

La bibliothèque "Enki" se charge de la simulation du robot dans un univers 2D (malgré que le viewer soit en 3D). Elle gère donc par exemple les calculs de la trajectoire d'un robot.

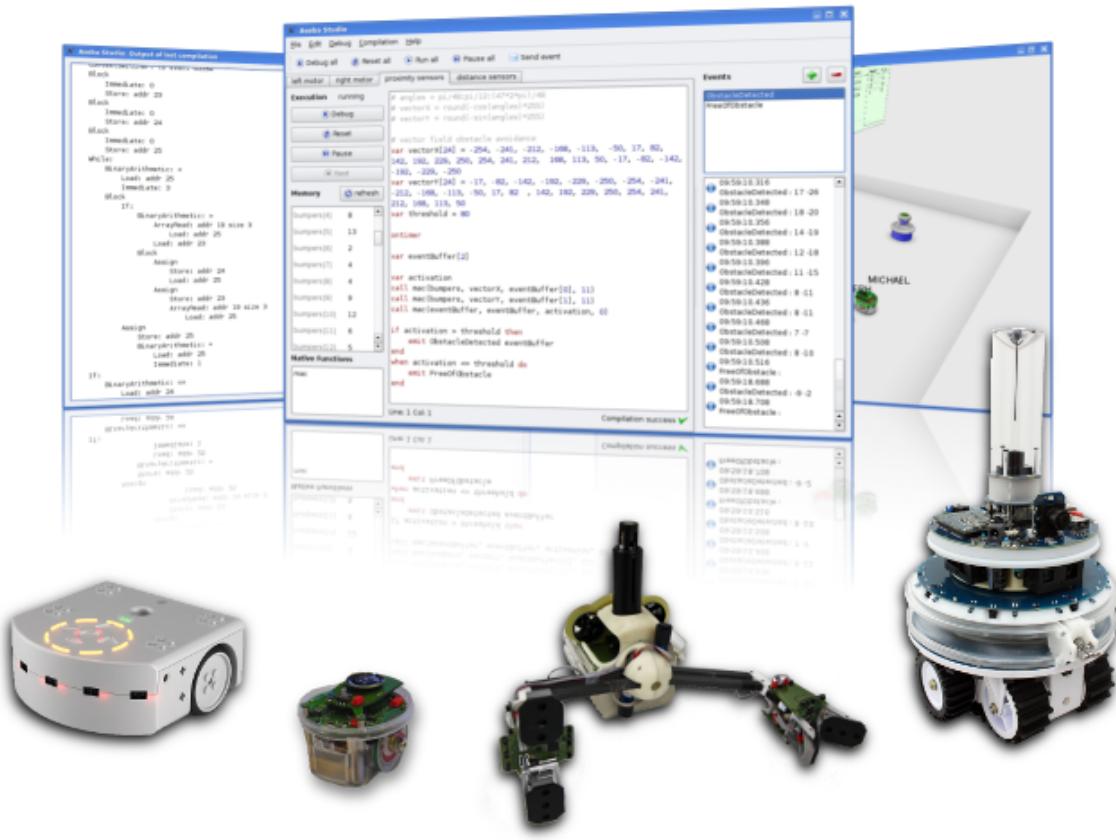


Figure 3: Interface du logiciel Aseba ainsi que divers robots programmables (Source : Wikidot Thymio)

Les différents robots programmables disponibles sur Aseba et présentés respectivement sur la figure 3 sont le Thymio II, l'e-puck, le hand-bot, le marXbot ainsi que l'Elisa-3 (qui ne figure pas sur l'image).

## Aseba Studio : interface détaillée

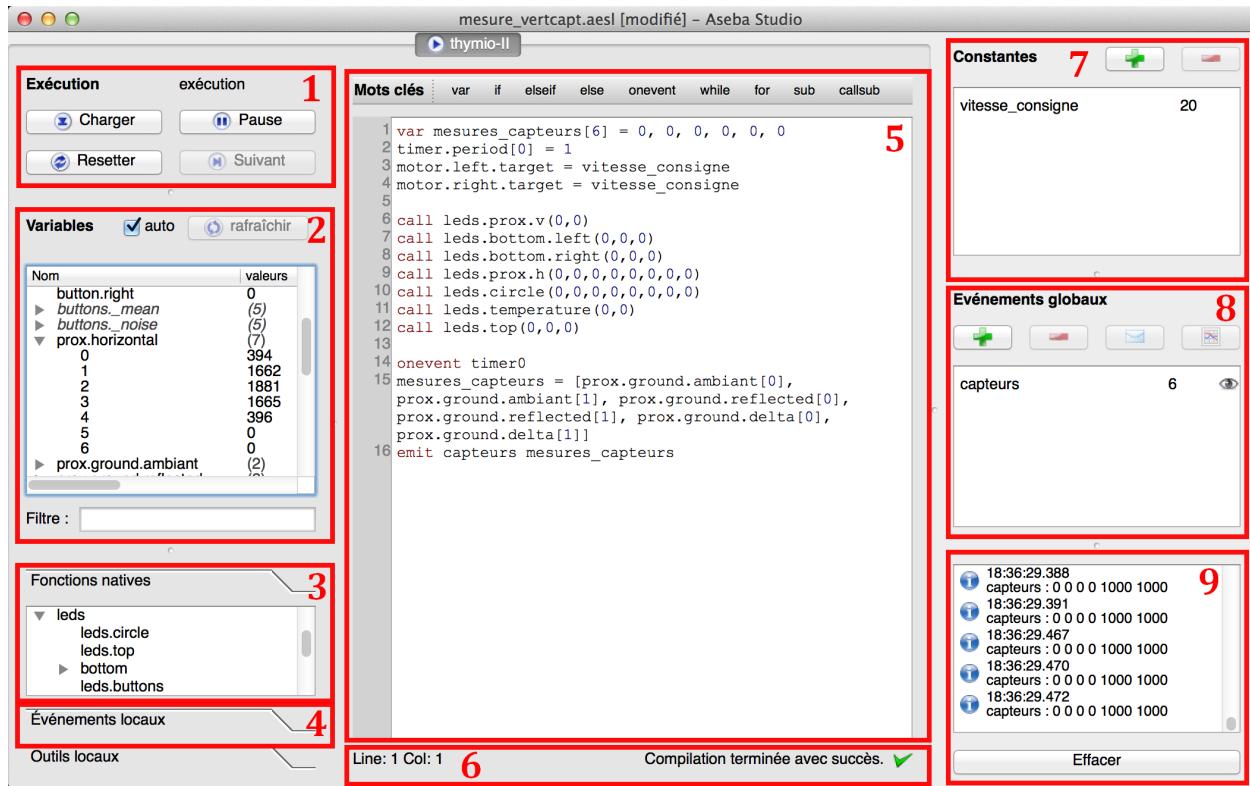


Figure 4: Interface détaillée du logiciel Aseba Studio

1. Interface de chargement, du lancement et de l'interruption du programme.
2. Fenêtre des variables du robot : elle contient toutes les variables informatives ainsi que les variables de contrôles. Il est possible de modifier une variable directement à travers cette fenêtre et d'en voir l'effet sur le robot.
3. La liste des fonctions natives du robot. Elles peuvent être appelées avec la balise "call" dans le code.
4. Les évènements locaux permettent avec la balise "onevent" d'exécuter un code défini pour un évènement particulier. Par exemple, l'évènement *motor* du Thymio II a lieu avec une fréquence de 10Hz, c'est à dire toutes les 10ms. Il est alors possible de mesurer certaines variables comme la vitesse des roues sur cet évènement.
5. Editeur du programme, c'est dans cette fenêtre qu'on édite ou qu'on peut observer l'exécution pas à pas du programme.
6. Débuggeur en temps réel : il va signaler directement à l'utilisateur si le code comprend une erreur de syntaxe et va la lui signaler dans cette fenêtre. Si elle n'en comprend pas, la compilation se fait automatiquement et le programme est prêt à être exécuté.
7. Fenêtre des constantes : on peut y définir certaines constantes qui peuvent être directement utilisables dans le code (exemple : *vitesse\_consigne* dans la fig. 4).
8. Les évènements globaux permettent de renvoyer un ensemble de valeurs (défini par le nombre de variables qu'on lui attribue) lorsqu'un évènement a lieu (exemple : *capteurs* dans la fig. 4).

9. Petite console qui renvoie les valeurs récupérées dans l'évènement global avec un certain timestamp.

### Connexion à un robot physique ou à un robot simulé

Au lancement du programme, Aseba demande l'accès à un robot à programmer.

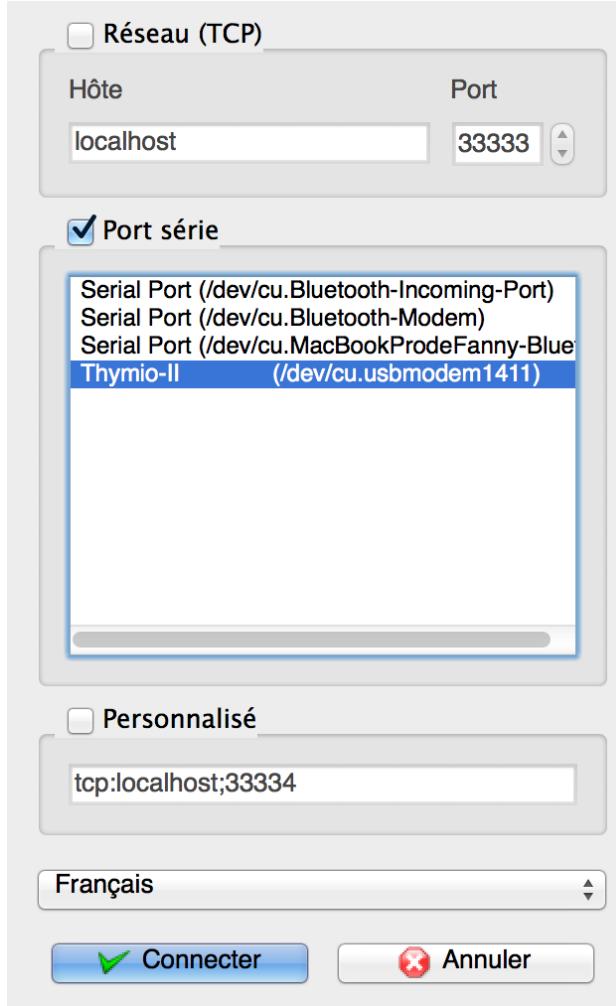


Figure 5: Fenêtre de connexion avec un robot

Il existe deux types de robots programmables qui puissent être connectés à Aseba :

- les robots physiques (et donc réels) connectés par USB ou Bluetooth par exemple.
- les robots simulés dans *Aseba Challenge* ou *Aseba Playground* à travers la bibliothèque *Enki*.

Seuls les robots *E-puck* et *Marxbot* sont simulés grâce à la bibliothèque *Enki* et peuvent être programmables sous Aseba à l'heure actuelle.

Le but de ce projet est de réaliser la simulation du robot *Thymio II* dans la bibliothèque *Enki* et de pouvoir le programmer avec l'interface *Aseba studio* et de le visualiser sous *Aseba Challenge*. Ainsi sa simulation peut permettre aux utilisateurs de programmer un robot Thymio II simulé sans devoir avoir recours à un robot physique.

### 1.3 Objectifs du projet : Donnée et travail demandé

*"Le projet consiste à implémenter un Thymio simulé dans un environnement similaire à Aseba Challenge ou Aseba Playground, qui proposent actuellement uniquement le robot e-puck. Il faudra principalement :*

- *Créer un modèle de Thymio simulé dans Enki (capteurs, déplacements, interactions)*
- *Créer un modèle de Thymio animé pour le viewer*
- *Adapter Aseba Playground pour pouvoir utiliser un Thymio simulé*
- *Créer un terrain de test dans Playground comprenant au moins une texture au sol avec piste à suivre et lignes à compter ainsi que des obstacles à éviter*

*Le modèle simulé devra reproduire au moins la gestion des moteurs de Thymio, les capteurs IR horizontaux et verticaux ainsi que l'affichage des LEDS. L'interface utilisateur vue depuis Aseba Studio sera identique à l'interface du Thymio réel, de façon à ce que les codes réalisés sur l'une ou l'autre plateforme soient compatibles.*

*Le code produit par l'étudiant pourra être distribué avec les prochaines versions du logiciel et devra donc être stable et rigoureusement testé. "*

### 1.4 Description du robot

Afin de pouvoir le simuler il est important d'énumérer les différents composants du robot Thymio II (ses capteurs, ses actuateurs ainsi que les supports interactif).

Les différents capteurs existants sur le robot et présentés en bleu sur la figure 6 sont :

- 7 capteurs de proximité orientés horizontalement (5 sont situés à l'avant du robot et 2 à l'arrière du robot)
- 2 capteurs au sol orientés verticalement
- 5 touches capacitatives
- 1 microphone pour entendre ou enregister des sons
- 1 capteur de température
- 1 capteur pour connaître le niveau de batterie restant
- 1 accéléromètre 3 axes permettant de détecter les chocs ou si le robot est incliné
- 1 récepteur de télécommande infrarouge

Les actuateurs du Thymio II sont :

- 1 haut-parleur
- 39 LED pour visualiser les capteurs actifs ou les différentes interactions du robot
- 2 roues contrôlées en vitesse (moteurs DC)

Enfin les différents supports du robot sont :

- 1 support crayon pour observer le déplacement du robot à l'aide d'un crayon
- 1 connectique USB nécessaire pour recharger et programmer le robot
- 1 lecteur de carte mémoire pour y stocker différents sons par exemple
- 1 bouton reset
- des prises mécaniques : fixations LEGO<sup>©</sup> et un crochet pour remorque

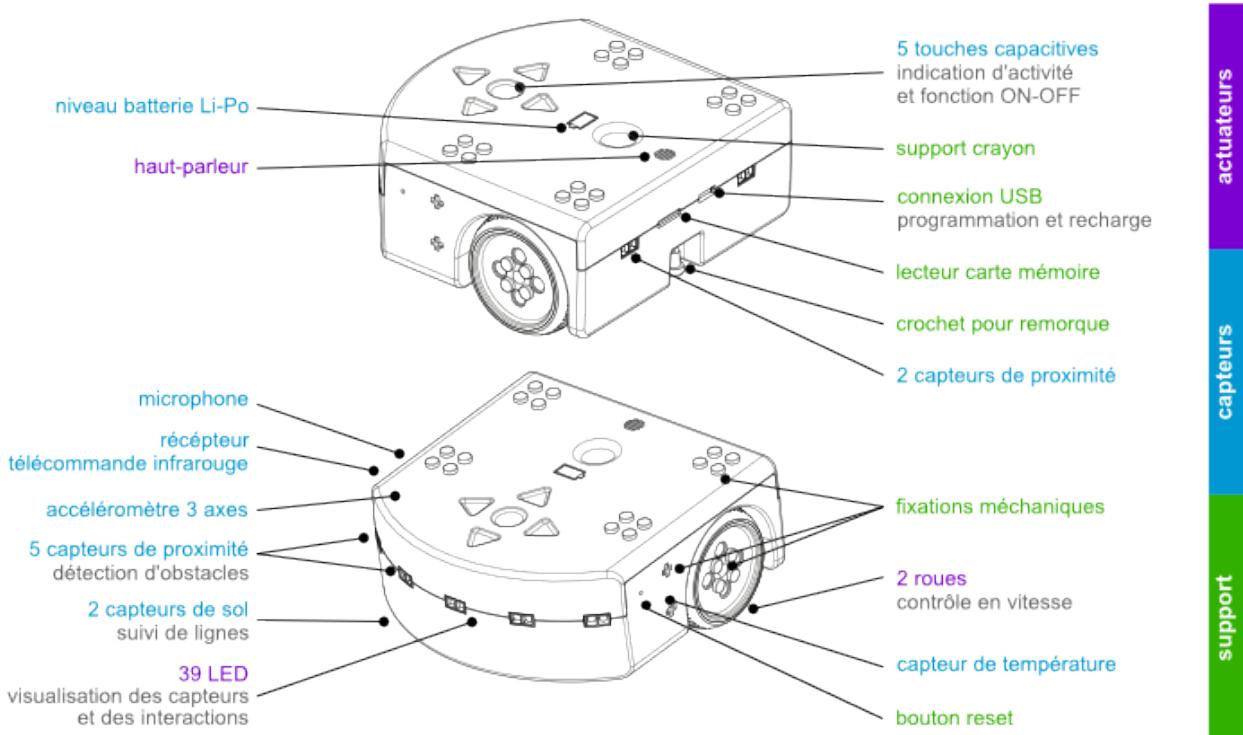


Figure 6: Emplacement des différents capteurs et actuateurs (Source : Wikidot Thymio)

Il existe pour le Thymio II des événements qui lui sont spécifiques et qui sont liés au divers capteurs du robot. Chaque événements est appelé sous certaines conditions ou avec une certaine fréquence. La liste de ces évènements est la suivante :

- L'évènement *prox* est appelé avec une fréquence de 10Hz
- L'évènement *motor* est appelé avec une fréquence de 100Hz
- L'évènement *acc* est appelé avec une fréquence de 16Hz
- L'évènement *temperature* est appelé avec une fréquence de 1Hz
- L'évènement *buttons* est appelé avec une fréquence de 20Hz
- L'évènement *rc5* est appelé lorsque le robot reçoit un signal (par télécommande)
- L'évènement *tap* est appelé lorsque le robot détecte un choc
- L'évènement *sound.finished* est appelé lorsque la lecture d'un son est terminée
- Les évènements *button.forward*, *button.left*, *button.backward*, *button.right* et *button.center* sont appelé lorsque les boutons désignés (respectivement avant, gauche, arrière, droite et centre) sont appuyé ou relâché.

## 1.5 Délimitation des objectifs du projet

Pour la simulation du robot Thymio II, certains de ces actuateurs ou capteurs peuvent être difficilement simulables ou ont peu d'intérêt de l'être pour l'utilisateur. De plus, la bibliothèque Enki permet la simulation d'un robot dans un univers 2D virtuel qui peut bloquer ou alors limiter l'utilisation de certains capteurs.

L'objectif du projet se résumera donc à la simulation des capteurs de proximités, des capteurs au sol, des moteurs ainsi que l'affichage des LEDs sur le robot.

Les autres capteurs et actuateurs ont été écartés du projet pour les raisons suivantes :

- Le *capteur de température* car il est peu utilisé et présente peu d'intérêt en simulation.
- L'*accéléromètre* perd de son intérêt car l'univers simulé est en 2 dimensions. Il peut cependant être envisageable pour la détection de choc.
- Le *microphone* et les *hauts-parleurs* n'ont pas non plus été choisis dans le cadre du projet car il n'existe pas de son réel dans l'univers simulé. On pourrait par contre permettre d'implémenter un module qui puisse lire des fichiers sons et les enregistrer par exemple. Une autre possibilité pourrait être de créer une connexion avec un micro et le logiciel de simulation.
- Les *5 touches capacitatives* sont aussi difficiles à implémenter puisque que l'utilisateur ne peut pas toucher le robot simulé. Une amélioration possible pourrait être de commander le robot à l'aide de touche du clavier ou encore d'implémenter la possibilité de cliquer sur les boutons du robot.
- Le *récepteur de télécommande infrarouge* présente peu d'intérêt en simulation et concernant la *batterie*, on a fait le choix d'avoir un robot qui ne se décharge pas en simulation pour ne pas avoir besoin de le recharger. On pourrait tout de même envisager, si nécessaire, de coder une variable qui diminue au cours du temps et qui stoppe l'utilisation du robot lorsqu'elle atteint 0.

Finalement, concernant les supports mécaniques et la connectique(USB et carte mémoire), il est évident qu'ils ne sont utilisables qu'avec le robot réel. L'ajout de matériel combinable serait trop long, trop complexe et sans grand intérêt concernant les réels objectifs du projet.

## 1.6 Matériel de test

Les mesures se sont effectuées à l'aide de 3 robots Thymio II (TH085, TH091 et TH095). Ils définissent donc un petit échantillon qui ne sera pas forcément représentatif de l'ensemble des robots Thymio II, mais qui pourra donner un bref aperçu de la dispersion des mesures de certains de ses composants ainsi que de mettre en évidence certaines mesures critiques : c'est-à-dire des composants défaillants ou encore les valeurs trop bruité renvoyés dans *Aseba*.

Les modèles proposés dans ce projet seront donc représentatifs de ces 3 robots afin d'être le plus similaire possible à l'ensemble des robots Thymio II.

Il est tout de même important de signaler que le robot simulé est principalement utilisé par des enfants, des professeurs ou par un public non initié à la robotique au niveau universitaire.

De plus, le matériel étant de très faible coût et dont la haute qualité n'étant pas l'objectif premier du robot, certaines mesures en particulier sur celles des moteurs sont très bruitées.

## 2 Modélisation des capteurs horizontaux

### 2.1 Caractéristique des capteurs

#### 2.1.1 Mesures et observations

Pour les mesures suivantes, les 3 robots ont été disposés devant un obstacle de couleur blanche (boîte d'emballage du Thymio II) à des distances variant entre 1cm et 18cm avec un pas de 1cm pour chacun des 7 capteurs sur 1 seconde et on a récupéré les données moyennées en unité aseba pour chaque capteur afin d'établir des graphiques tels que sur la figure 8 (mesure sur le robot Thymio II TH095) pour chaque robot (voir Annexe G).

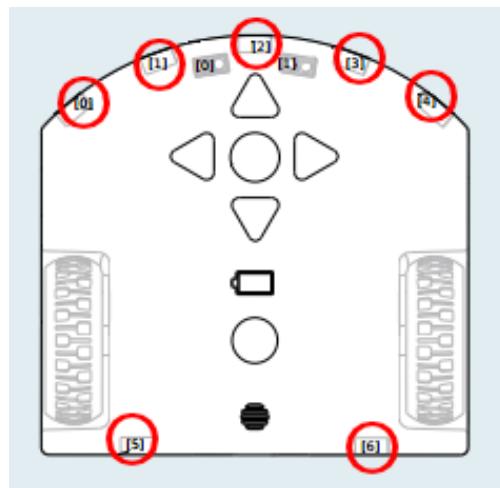


Figure 7: Emplacement des 7 capteurs horizontaux sur le robot Thymio II

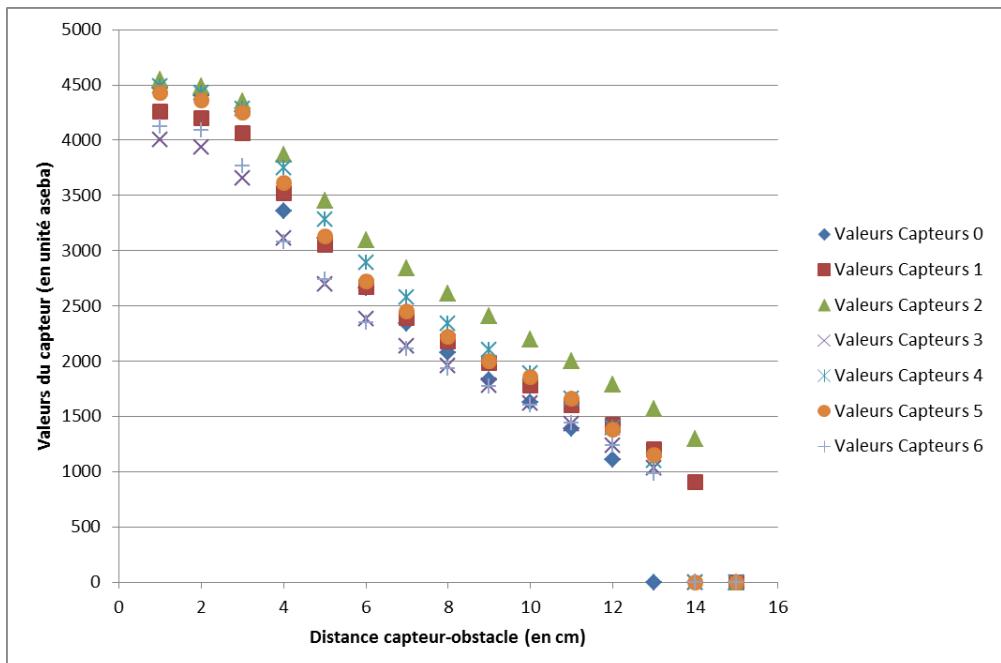


Figure 8: Mesure de la valeur des capteurs (en unité aseba) du robot Thymio II TH095 en fonction de la distance (en cm)

On peut observer tout d'abord que le capteur ne renvoie plus d'information à partir d'une certaine distance. La figure 8 montre que la distance maximum varie entre 13 et 15 cm pour le robot TH095 (elles varient entre 13 et 18 pour l'ensemble des 3 robots).

Les valeurs diminuent non linéairement en fonction de la distance et on peut observer une certaine saturation du capteur lorsque la distance est trop faible, c'est-à-dire une plus faible diminution de la valeur des capteurs sur les 3 premiers centimètres environ. Ce qui peut s'expliquer par le fait que plus la distance est faible plus le capteur peut détecter les rayonnements infrarouges reflétés sur l'obstacle, mais il est cependant limité par ce qu'il peut recevoir : les récepteurs saturent.

### 2.1.2 Modélisation et choix du modèle

Concernant le modèle caractéristique, on reprend un modèle déjà existant (pour les capteurs IR de l'e-puck) et définit selon l'équation suivante :

$$valeurs = f(x) = \frac{m \cdot (c - x_0^2)}{x^2 - 2 \cdot x_0 \cdot x + c} \quad (1)$$

avec  $m$  la valeur maximale du capteur en ua (unité aseba),  $x_0$  la distance minimale en cm à laquelle est renvoyée la valeur maximale,  $x$  la distance en cm de l'obstacle par rapport au capteur et  $c$  un paramètre de calibration.

Sous Matlab (voir Annexe A), les paramètres  $x_0$ ,  $c$  et  $m$  ont été optimisés pour chaque capteur (19 en tout, 2 ont été écartés à cause de leur valeurs déviante : les capteurs 1 et 21) afin que l'erreur quadratique entre le modèle caractéristique et les valeurs réelles soit minimale. De plus, afin d'optimiser le temps de calcul, on a réduit les plages de valeurs pour chaque paramètre :

- Le paramètre  $m$ , soit la valeur maximale renvoyé par le capteur, est défini entre 4000ua et 5000ua pour un pas de 1. En effet les 21 capteurs ont tous renvoyé une valeur maximale dans cette plage là, il était astucieux de réduire la plage de données en conséquence.
- Concernant le paramètre  $x_0$ , les valeurs maximales sont atteintes en dessous du centimètre mais sont difficiles à déterminer : la plage de données choisie est entre 0cm et 1cm avec un pas de 0.1cm.
- Pour terminer, le choix d'un intervalle de calcul pour le paramètre de calibration  $c$  n'était pas déterminable trivialement. Au départ, on a choisi une plage large (de 0 à 5000 au départ) avec un grand pas (de 5 au départ) et les valeurs calculées se situaient entre 40 et 110. La plage s'est donc réduite de 1 à 120 pour un pas de 0.1.

Sous matlab, on obtient un tableau de valeurs (voir annexe B) qui correspondent aux 3 paramètres optimaux pour chaque capteur. Le modèle final est calculé avec la valeur moyenne de chacun de ces paramètres :  $x_0 = 0.03cm$ ,  $m = 4505ua$  et  $c = 73.29$ . Et on obtient une courbe caractéristique qui suit bien la réalité comme présenté sur la figure 9.

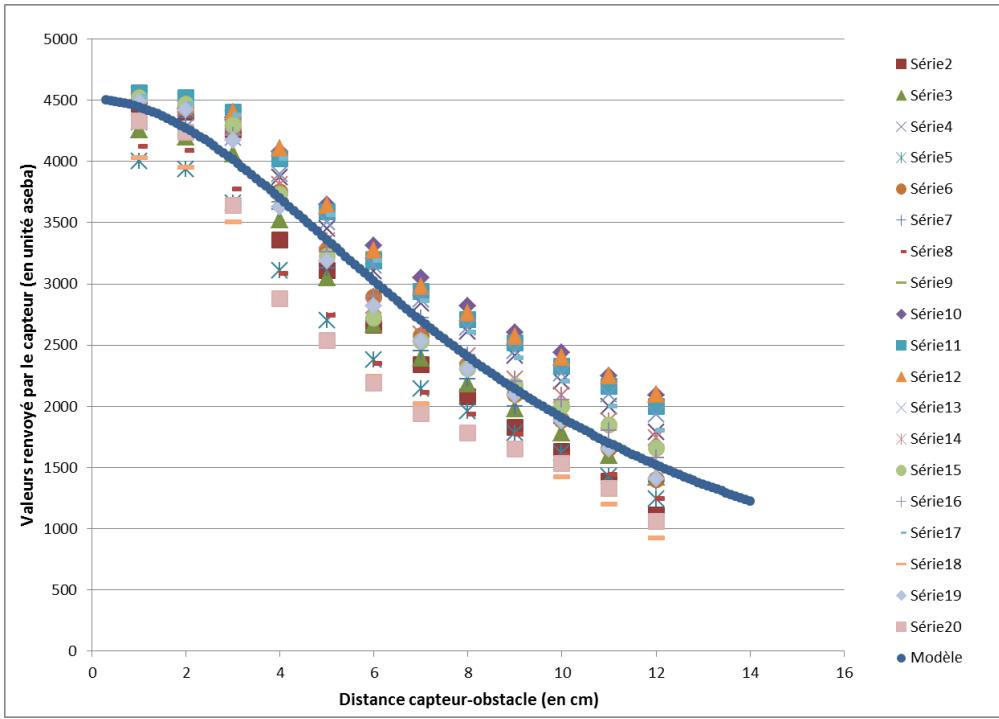


Figure 9: Caractéristique de la valeur des capteurs (en unité aseba) en fonction de la distance (en cm)

Concernant la portée du capteurs, on a relevé dans le tableau suivant la distance maximale à laquelle chaque capteur renvoie une information :

Table 1: Portée en cm des 7 capteurs pour chaque robot

Robot	TH085	TH091	TH095
Capteur 0	12	15	12
Capteur 1	14	16	14
Capteur 2	15	16	14
Capteur 3	12	18	15
Capteur 4	13	16	13
Capteur 5	13	14	13
Capteur 6	10	14	15

La portée choisie du capteur du modèle de simulation est de **14cm** c'est à dire la moyenne de ces 21 valeurs.

### Amélioration possible

Dans le cadre de ce projet, le choix a été fait de simuler un seul modèle de capteur avec les paramètres définis précédemment. Cependant, il est tout à fait possible d'intégrer un paramètre de dispersion qui permettraient de donner des valeurs aléatoires dans une certaines plages de données pour les paramètres  $c$ ,  $x_0$ ,  $m$  ainsi que la portée d'un capteur.

## 2.2 Caractéristique du bruit des capteurs

### 2.2.1 Mesures et observations

Pour ces mesures, on a utilisé 3 capteurs (les capteurs 2 de chaque robot), et on a mesuré sur 10 secondes le bruit de chaque capteur à des distances variantes entre 1cm et 16cm avec un pas de 1cm à l'aide de l'évènement "prox" (ayant une fréquence de 10Hz : on obtient 10 valeurs par seconde). Ces mesures sont représentées sur la figure 10.

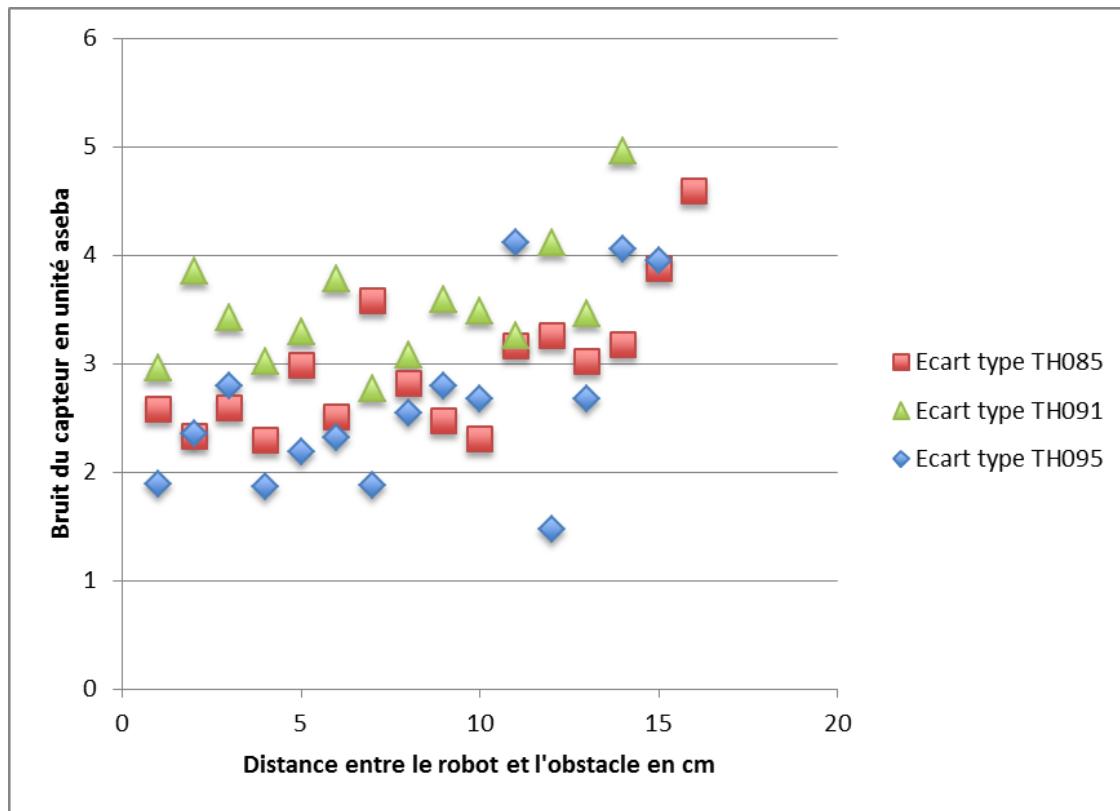


Figure 10: Mesure du bruit des capteurs(en unité aseba) en fonction de la distance (en cm)

Dans un premier temps, on peut constater que le bruit reste très faible. Le bruit (ou l'écart type) du signal varie entre 2 et 4 ua (unité aseba) environ contre une moyenne variant entre 4500ua (au maximum) et 1200ua (à la distance limite).

De plus on constate que l'écart type est indépendant de la distance. Il a cependant une tendance à augmenter légèrement (30% ou environ 1ua) aux valeurs limites.

### 2.2.2 Modélisation et choix du modèle

Pour le modèle de simulation, un bruit indépendant de la distance sera ajouté sur le modèle caractéristique et ayant pour valeur la moyenne des écarts types des 45 signaux.

Table 2: Bruit (ou écart type) exprimé en ua (unité aseba) des capteurs n°2 des 3 robots

Distance	TH085	TH091	TH095
1cm	2.578	2.960	1.886
2cm	2.334	3.858	2.349
3cm	2.586	3.427	2.793
4cm	2.288	3.023	1.861
5cm	2.982	3.288	2.186
6cm	2.506	3.779	2.321
7cm	3.581	2.773	1.873
8cm	2.817	3.084	2.542
9cm	2.470	3.595	2.795
10cm	2.305	3.488	2.674
11cm	3.157	3.255	4.110
12cm	3.252	4.113	1.477
13cm	3.015	3.459	2.677
14cm	3.178	4.965*	4.061*
15cm	3.880*	-	3.944*
16cm	4.592*	-	-

On obtient un bruit  $\sigma = 2.868ua$ . Cette moyenne s'est calculée sans prendre en compte les écarts types des signaux pour les valeurs limites (signalé par un "\*" dans le tableau 2).

### Amélioration possible

On pourrait reproduire le phénomène d'augmentation du bruit aux valeurs limites. Par exemple, si la distance entre le capteur et l'obstacle est à moins de 2 cm de la portée maximale du capteur, on peut ajouter un autre paramètre (voire une fonction du 1er par exemple) qui viendra s'ajouter au bruit pour le signal aux valeurs limites. Les tests se sont effectués seulement sur un obstacle de couleur blanche, mais on pourrait aussi simuler la sensibilité à la couleur des capteurs horizontaux.

### 3 Modélisation des moteurs DC

Pour les mesures suivantes, on a imposé des vitesses de consignes variant entre 50ua et 600ua avec un pas de 50ua pendant 1 seconde pour chaque robot (c'est à dire pour les 6 moteurs). Ainsi on a pu récupérer les informations suivantes : les vitesses réelles, le PWM et le courant pour chaque moteur. L'acquisition des données se fait grâce à l'événement "motor" (dans aseba) dont la fréquence est de 100Hz, ainsi on récupère 100 valeurs par seconde. Dans un premier temps, on s'intéressera à la vitesse des moteurs.

#### 3.1 Vitesse des moteurs

##### 3.1.1 Détermination de la vitesse maximale des moteurs

###### Mesures et observations

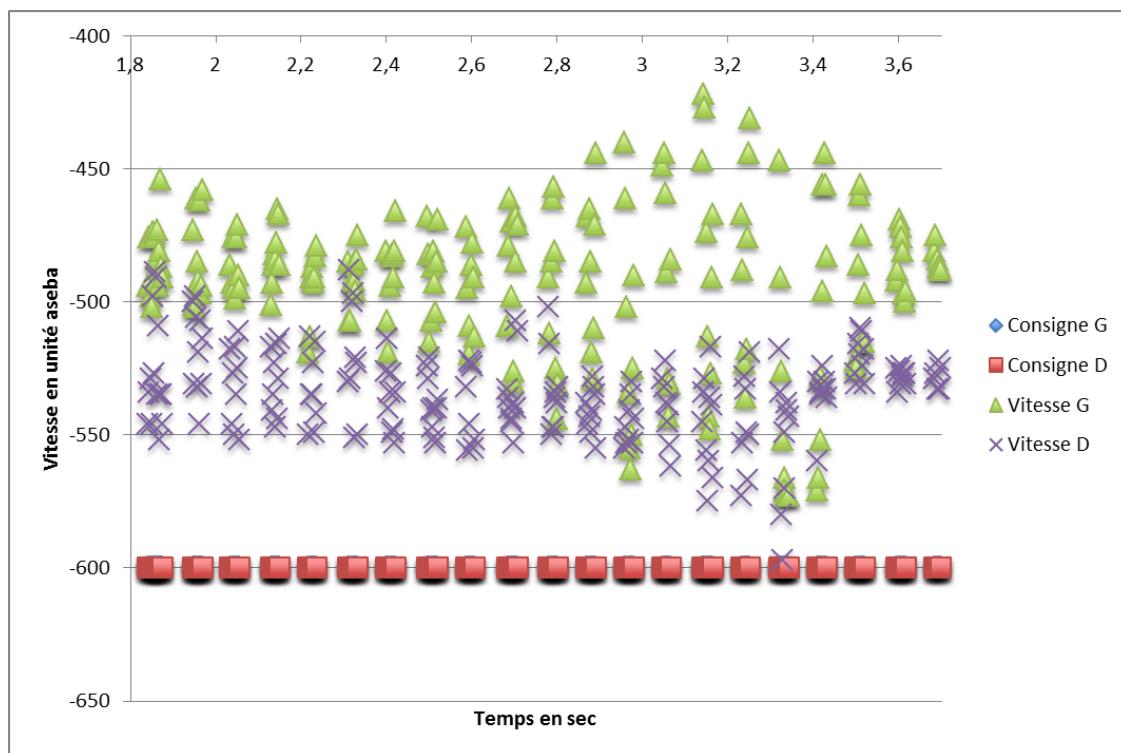


Figure 11: Mesure de la vitesse maximale des moteurs (en ua) du TH085 en fonction du temps (en sec) pour une vitesse de consigne de -600ua

Le robot plafonne effectivement à une vitesse absolue d'environ 500ua avec un bruit non négligeable. Pour chaque moteur, on a imposé une vitesse de consigne supérieure à la valeur limite du robot dans les deux sens (avant et arrière). Et on a calculé leur vitesse maximale en moyennant le signal renvoyé sur aseba pendant un temps donnée (environ 2 secondes). On obtient les valeurs suivantes :

Table 3: Portée en cm des 7 capteurs pour chaque robot

Robot	Roue Gauche		Roue droite		
	Sens	Avant	Arrière	Avant	Arrière
TH085	494.86	-490.68	520.81	-532.73	
TH091	490.17	-462.37	468.44	-473.45	
TH095	515.88	-517.57	518.53	-524.36	

On peut aussi remarquer que la vitesse maximale varie en fonction des moteurs : cela s'explique par niveau de la batterie, de la surface ou de la puissance non exacte du moteur.

### Modélisation et choix du modèle

La moyenne de ces valeurs maximales donne exactement 500.81 ce qui se trouve être la vitesse maximale conseillée pour l'utilisation du Thymio II. La vitesse maximale choisie pour le modèle simulé est donc de 500ua.

**Remarque :** A 500ua on a mesuré approximativement en combien de temps chaque robot parcourait 1m. Le résultat obtenu est de 16.6cm/s et qui sera considéré dans le modèle simulé comme l'équivalence en vitesse des 500ua.

#### 3.1.2 Caractéristique du profil de vitesse des moteurs

##### Mesures et observations

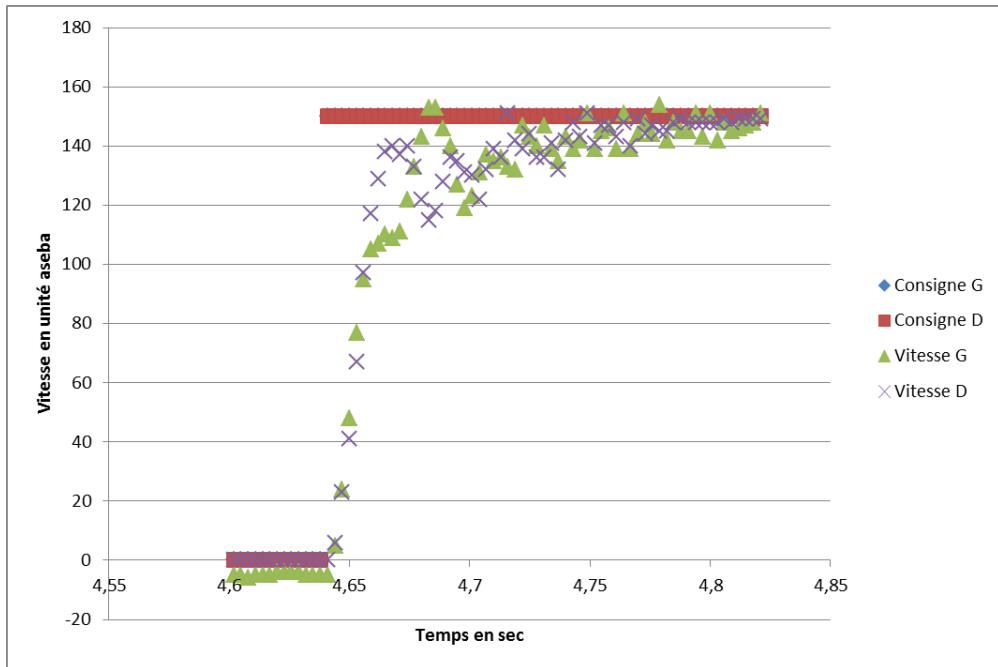


Figure 12: Profil de la vitesse (en unité aseba) en fonction du temps (en seconde)

On peut remarquer sur la figure 12, que l'augmentation de la vitesse n'est pas instantanée et que le profil de vitesse n'est pas linéaire. Le profil trouve de nombreuses similarités avec le profil de la tension dans un circuit RC.

### Modélisation et choix du modèle

On a choisi une fonction exponentielle pour décrire la caractéristique du profil de vitesse avec une certaine constante de temps (ou retard)  $\tau$  :

$$v(t) = v_{max} \cdot (1 - e^{-\frac{t}{\tau}}) \quad (2)$$

Pour chaque vitesse imposée et pour chaque moteur, on a calculé la constante de temps  $\tau$  en seconde (voir annexe C) en le déterminant graphiquement (sous excel) : en calculant au bout de quel temps, le moteur atteint la vitesse correspondant à  $0.63 \cdot v_{max}$ . Le tau résultant est calculé en moyennant toutes les valeurs obtenues :

$$\tau = 0.025 \text{ sec} = 25 \text{ ms} \quad (3)$$

On obtient ainsi le modèle suivant :

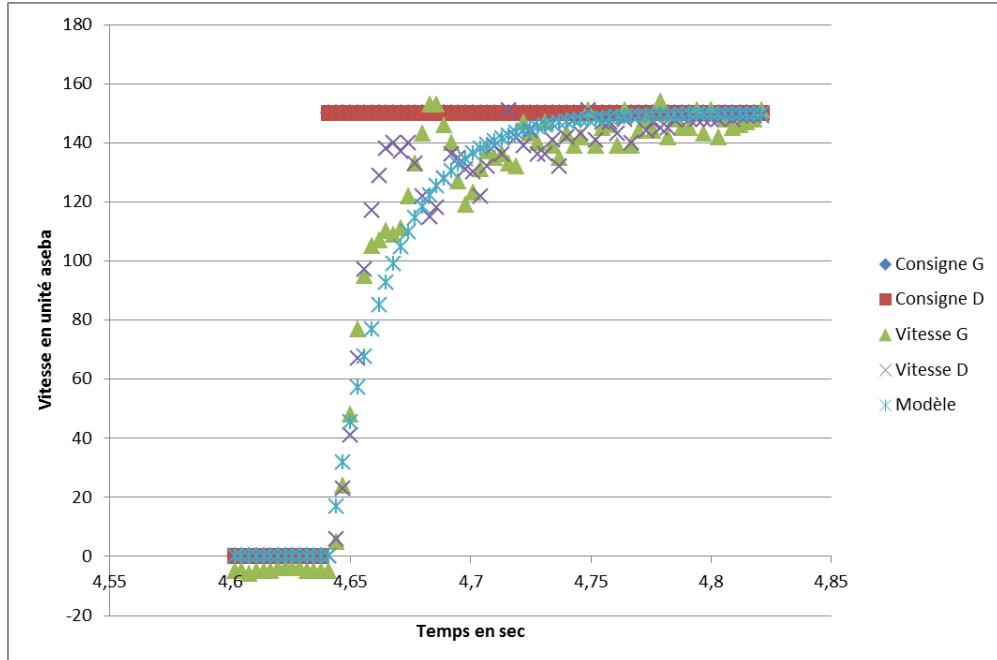


Figure 13: Modèle de profil de vitesse proposé pour la simulation (en unité aseba) en fonction du temps (en seconde)

### 3.1.3 Caractéristique du bruit des moteurs

#### Mesures et observations

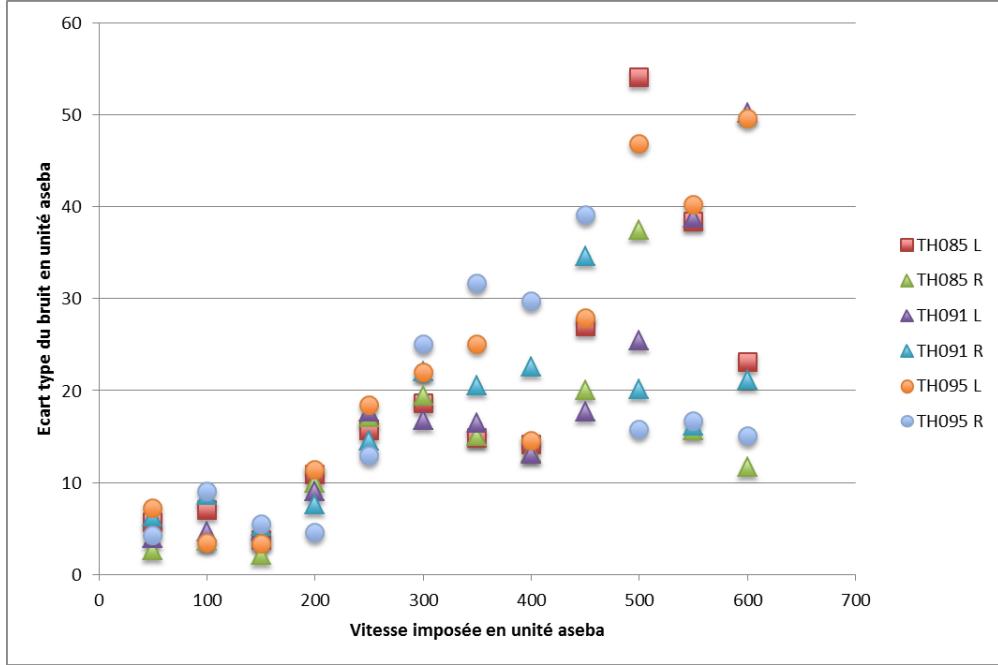


Figure 14: Bruit des moteurs (écart type des données) exprimé en ua en fonction de la vitesse imposée (en unité aseba)

Pour ces mesures on a calculé l'écart type sur les mêmes données effectuées précédemment (après avoir calculé la moyenne). On constate que le bruit a tendance à augmenter en fonction de la vitesse des roues. On a un bruit très important :  $\sigma \approx 50\text{ua}$  pour des vitesses en moyenne à 500ua. Le profil du bruit reste à peu près similaire entre les différents moteurs pour des vitesses inférieures à 300ua, cependant elles se dispersent fortement au dessus de 300ua.

#### Modélisation et choix du modèle

on a choisi un modèle caractéristique du premier degré pour le bruit. Tout d'abord, on a établi les 6 droites de tendance (avec Excel) représentant l'augmentation du bruit  $\sigma_{bruit}$  de chaque moteur en fonction de la vitesse  $v$  des moteurs (voir fig. 15) :

- TH085 L :  $\sigma_{bruit} = 0.0628v - 0.9783$
- TH085 R :  $\sigma_{bruit} = 0.0342v + 2.9189$
- TH091 L :  $\sigma_{bruit} = 0.0692v - 4.3017$
- TH091 R :  $\sigma_{bruit} = 0.0346v + 5.3975$
- TH095 L :  $\sigma_{bruit} = 0.0823v - 4.2584$
- TH095 R :  $\sigma_{bruit} = 0.0342v + 6.3112$

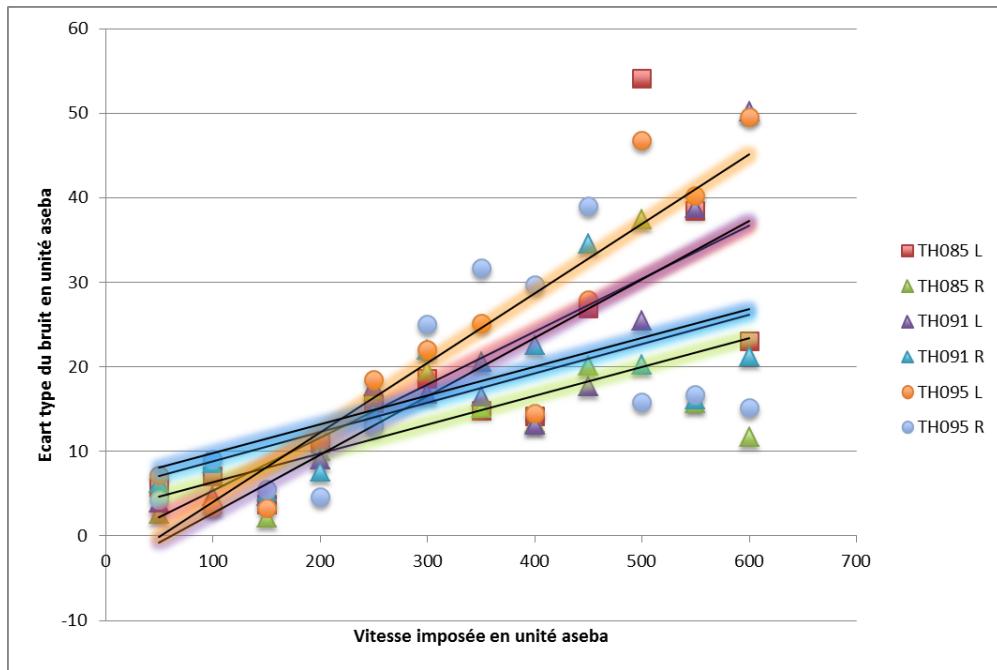


Figure 15: Bruit des moteurs (écart type des données) exprimé en ua avec leur droite caractéristique en fonction de la vitesse imposée (en unité aseba)

Et dans un second temps j'ai établi la moyenne des 6 droites pour établir un modèle linéaire pour la caractéristique du bruit sur la vitesse des moteurs.

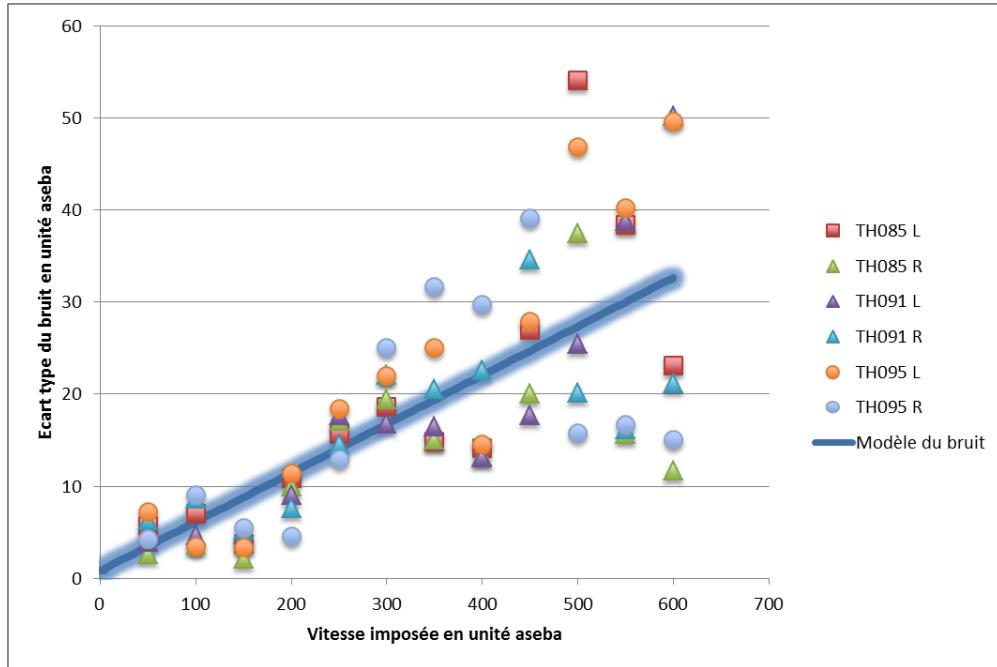


Figure 16: Modèle linéaire du bruit exprimé en ua en fonction de la vitesse imposée (en unité aseba)

Le modèle résultat est donné par l'équation suivante :

$$\sigma_{bruit} = 0.053v + 0.85 \quad (4)$$

avec  $v$  la vitesse des moteurs.

## 3.2 Le PWM

### 3.2.1 Caractéristique du PWM

Mesures et observations

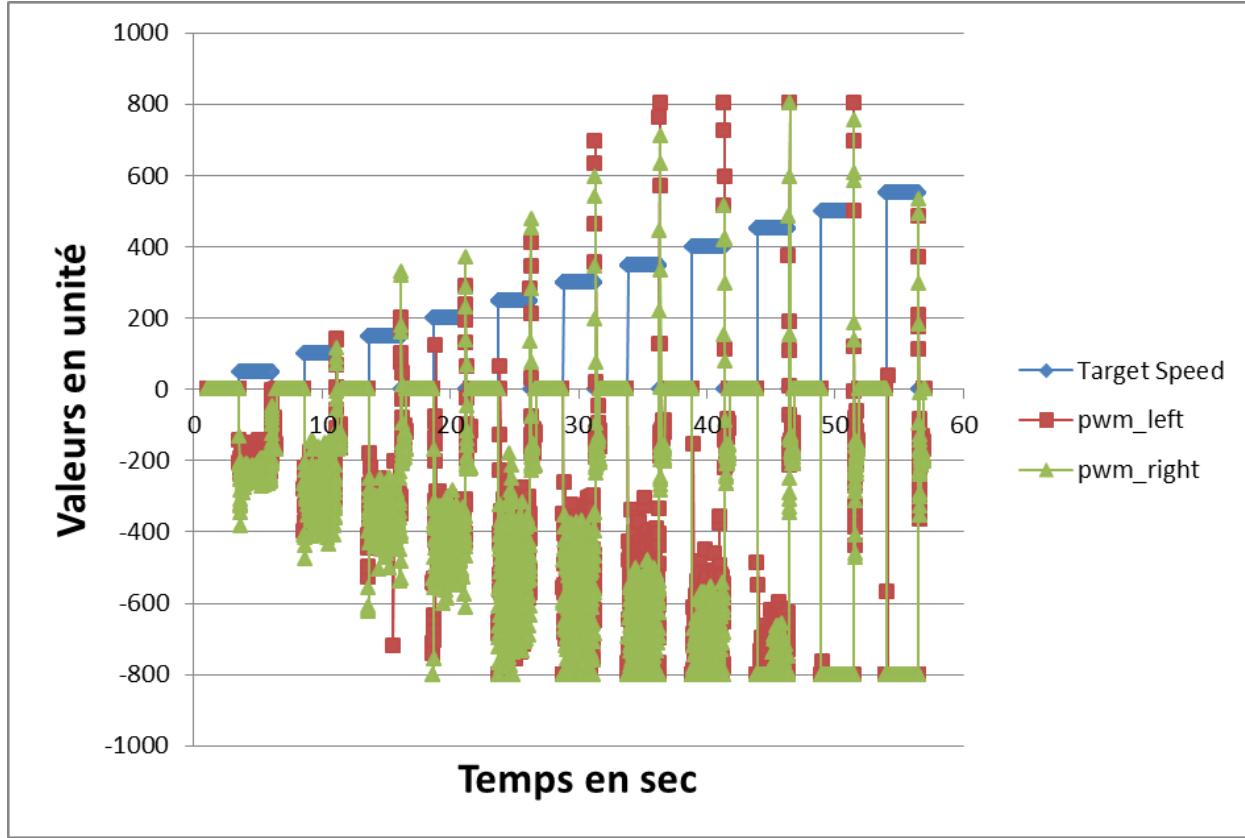


Figure 17: Valeurs du PWM (en ua) en fonction de la vitesse (en ua)

Le PWM est exprimé entre une valeur de 0 à -801 pour des vitesses de consigne positives et entre 0 et 801 pour des vitesses négatives. D'après la figure 17, on peut remarquer que la valeur moyenne du PWM tend à augmenter avec la vitesse des moteurs et les valeurs de PWM sont fortement bruitées. De plus, il sature à la valeur -801 (ou 801 en fonctionnement inverse).

On peut déjà en conclure que le rapport cyclique de la PWM situé en unité aseba entre 0 et 801 (équivalent au rapport cyclique entre 0 et 1).

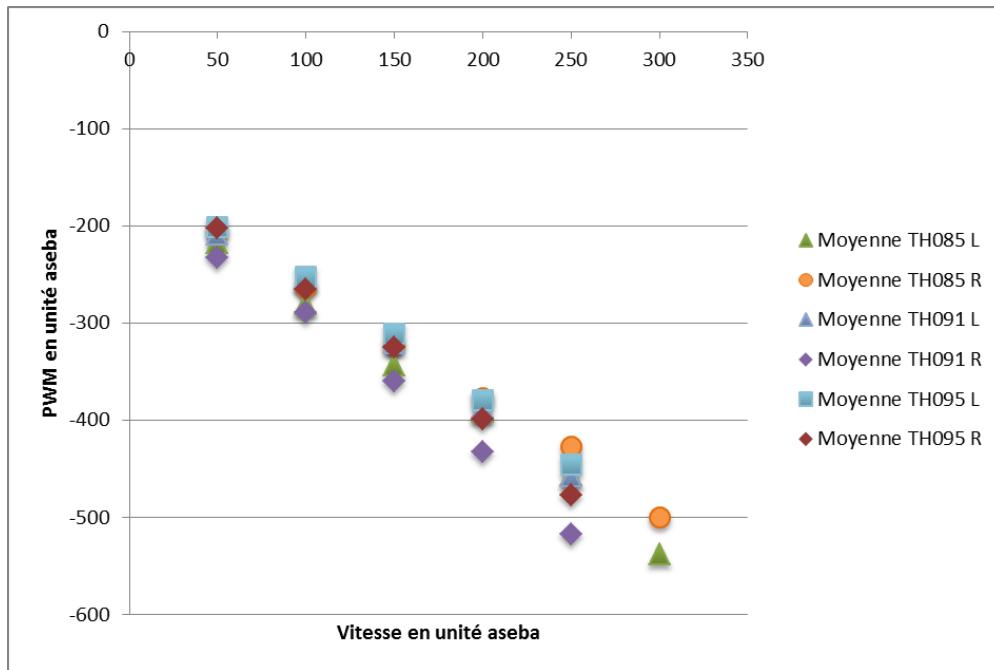


Figure 18: Valeurs du PWM en moyenne pour (en ua) en fonction de la vitesse (en ua)

On remarque aisément que la valeur absolue du PWM augmente linéairement en fonction de la vitesse des moteurs.

### Modélisation et choix du modèle

Pour le modèle du PWM, on a effectué une démarche similaire à celle effectuée pour la caractéristique du bruit des moteurs, c'est à dire que j'ai calculé les droites de tendance de chaque moteur (sous Excel) afin de déterminer la droite moyenne résultante des 6 autres (voir figure 18)

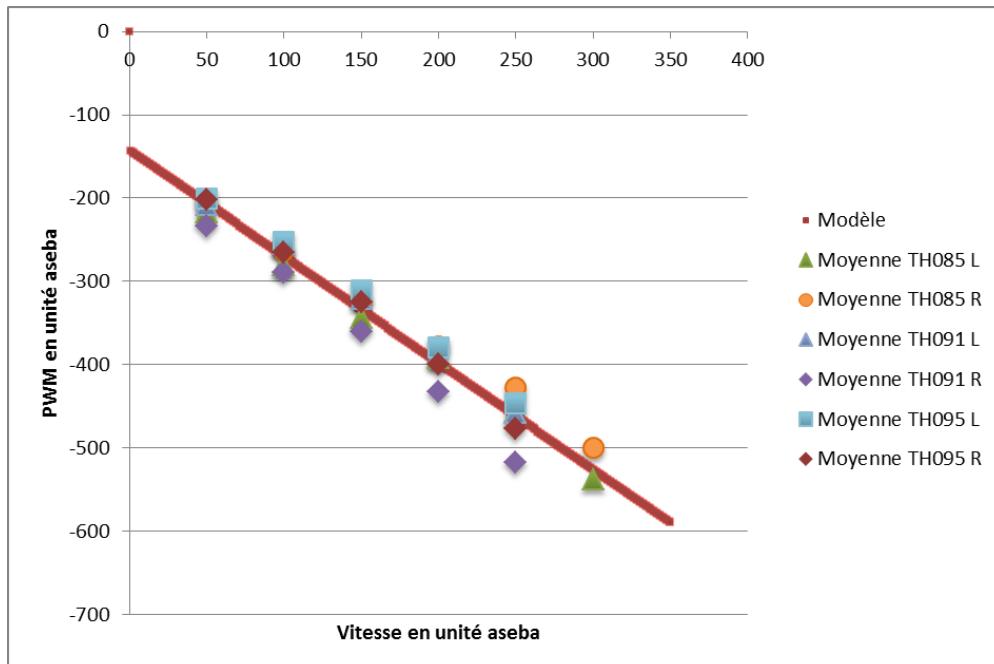


Figure 19: Valeurs du PWM en moyenne pour (en ua) en fonction de la vitesse (en ua)

La droite caractéristique de la figure ci-dessus se note selon l'équation suivante :

$$PWM = -1.2806 \cdot v - sgn(v) \cdot 142.14 \quad (5)$$

### Remarques :

- le signe de  $v$  ( $sgn(v)$  dans la formule) permet de calculer des PWM positifs pour les vitesses négatives (fonctionnement en sens inverse des moteurs).
- le profil du PWM est similaire à celui du profil de vitesse puisqu'il est dépendant de la vitesse. Les retards  $\tau$  correspondants sont donc les mêmes que pour ceux de la vitesse (voir Annexe C).
- le rapport cyclique correspondant peut se calculer en pourcentage en multipliant le résultat final par  $\frac{100}{801}$ .

### 3.2.2 Caractéristique du bruit du PWM

#### Mesures et observations

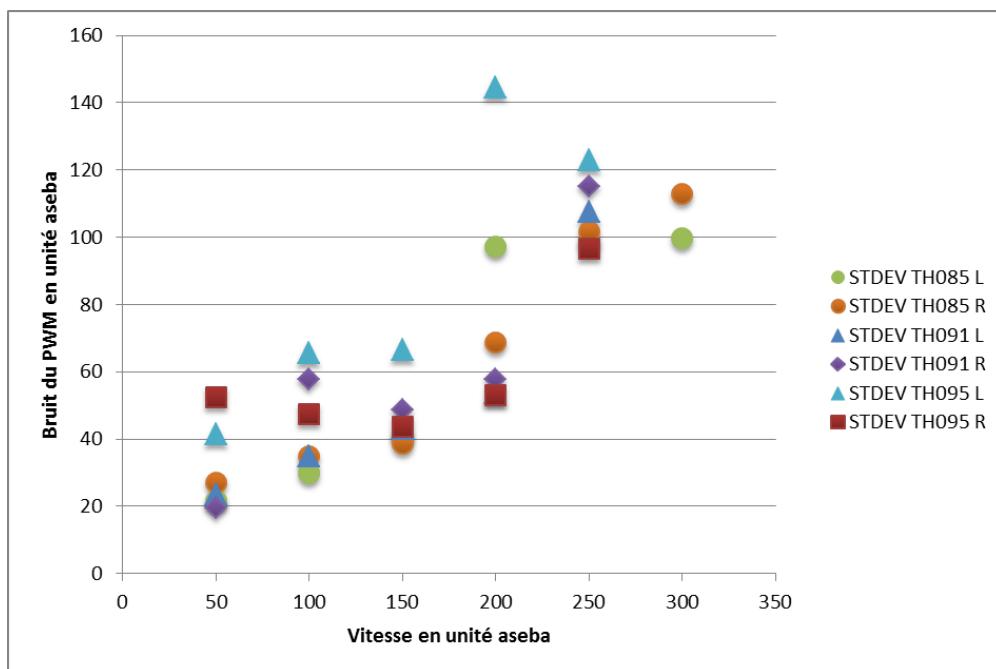


Figure 20: Bruit du PWM (écart type) exprimé en ua en fonction de la vitesse (en ua)

D'après la figure 20, on constate que le bruit (ou l'écart type) du PWM augmente en fonction de la vitesse et est assez important si on le compare à la moyenne. Par exemple, pour une vitesse des moteurs fixés à  $100\text{ua}$ , on obtient une moyenne du PWM d'environ  $-270.2$  avec un bruit  $\sigma \approx 43.3$ .

## Modélisation et choix du modèle

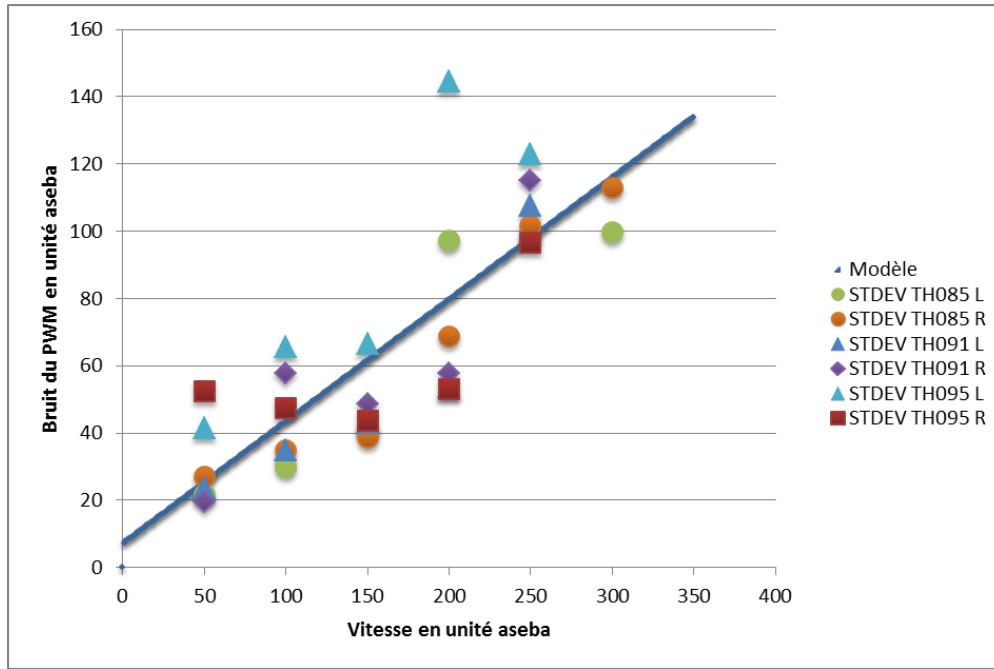


Figure 21: Comparaison entre le modèle et les mesures du bruit du PWM (écart type) exprimé en ua en fonction de la vitesse (en ua)

Selon une méthode similaire à l'établissement de la caractéristique du bruit des moteurs, j'ai établi le modèle du bruit selon une fonction du 1er degré établi sur la figure 21 et elle est définie selon l'équation suivante :

$$\sigma_{bruitPWM} = 0.3629 \cdot v + 7.01 \quad (6)$$

### 3.3 Le courant

#### 3.3.1 Courant maximum $i_{max}$

##### Mesures et observations

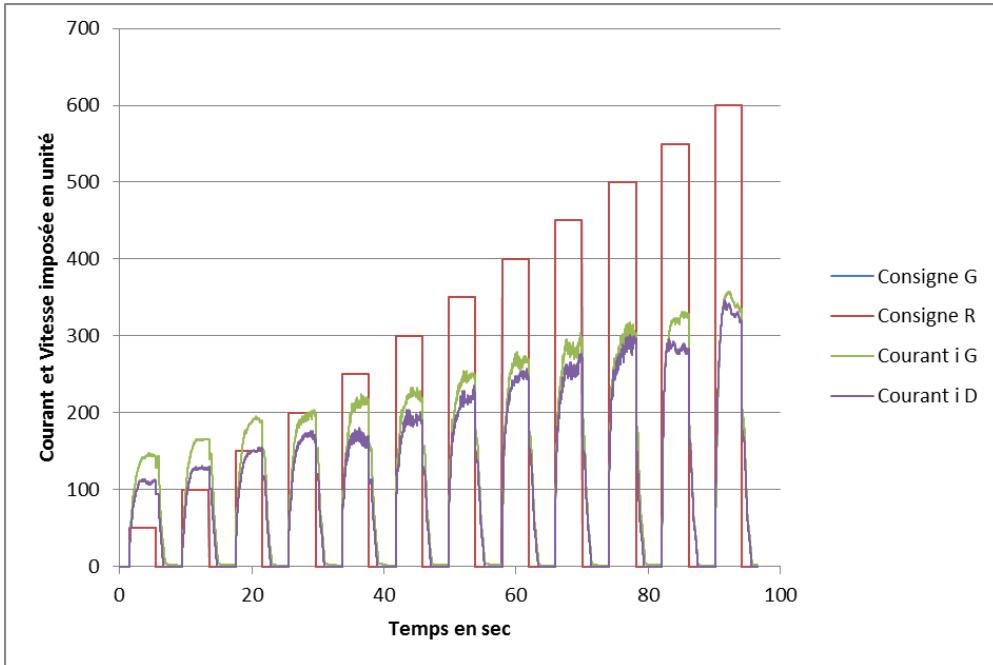


Figure 22: Mesure du courant (en unité aseba) en fonction du temps (en seconde) pour des vitesses de consignes différentes

D'après la figure 22, on constate que le courant augmente en fonction de la vitesse : ce qui s'explique logiquement car le moteur est à courant continu et donc le couple est proportionnel à  $I$ . Concernant le profil du courant, on est discutera dans la partie 3.3.2.

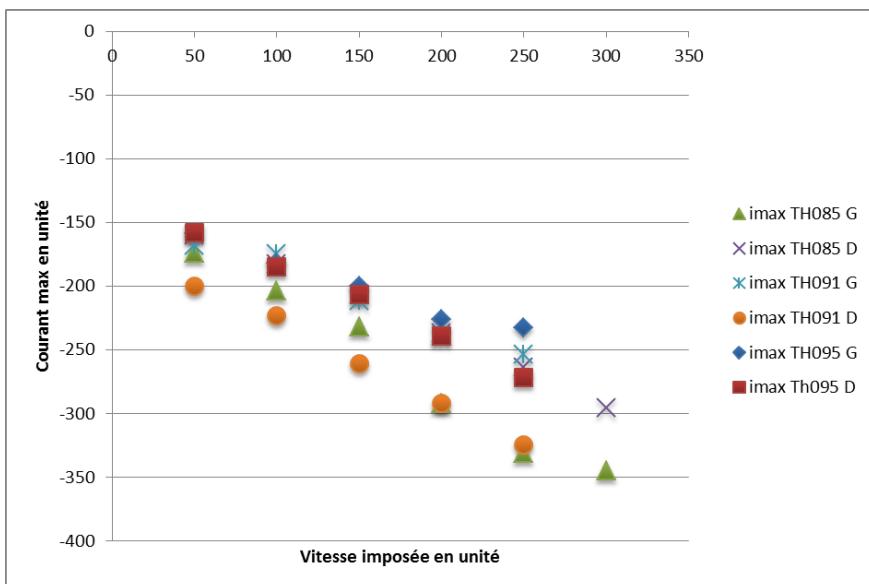


Figure 23: Mesure du courant moyen (en unité aseba) en fonction de la vitesse (en unité aseba)

Les valeurs de la figure 23 ont été calculées après avoir effectué la moyenne des courants maximaux en fonction de chaque vitesse des moteurs (avec un pas de 50ua). La figure précédente confirme ainsi que le courant maximal  $i_{max}$  est proportionnel à la vitesse.

### Modélisation et choix du modèle

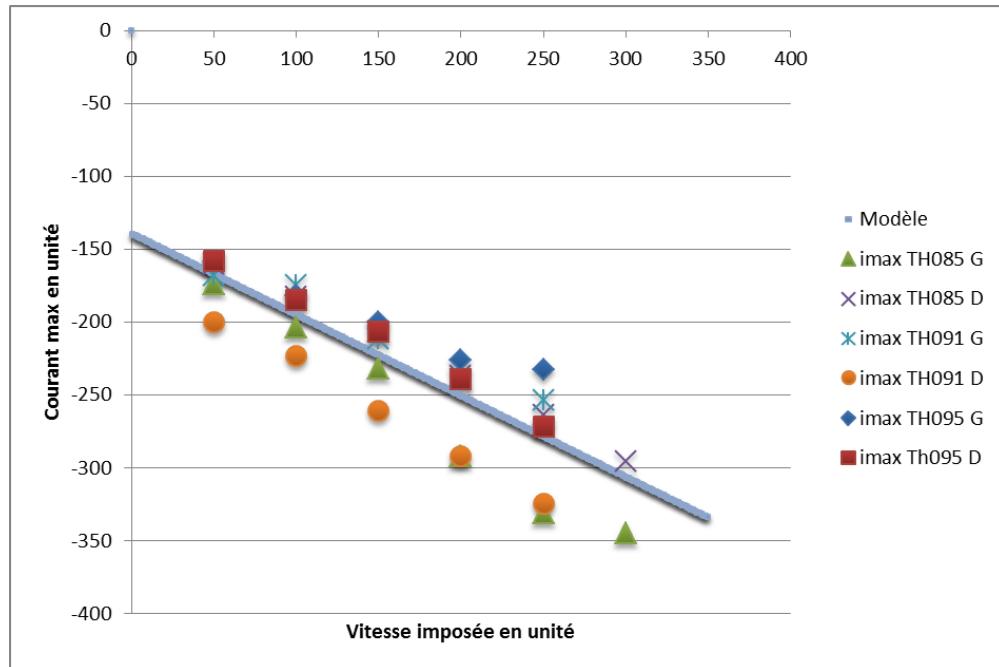


Figure 24: Comparaison entre le modèle et le courant mesuré (en unité aseba) en fonction de la vitesse (en unité aseba)

Suite aux observations faites précédemment et en effectuant la même méthode qui a été faite dans les sections précédentes (moyenne des droites de tendance sous excel), un modèle linéaire a été choisi pour la caractéristique du courant maximal et est défini par l'équation suivante :

$$i(v) = -0.5572 \cdot v - \text{sgn}(v) \cdot 139.32 \quad (7)$$

**Remarque :** similairement au PWM, le signe de v ( $\text{sgn}(v)$  dans la formule) permet de calculer le courant de signe positif pour les vitesses négatives (fonctionnement en sens inverse des moteurs).

### 3.3.2 Caractéristique du courant i

#### Mesures et observations

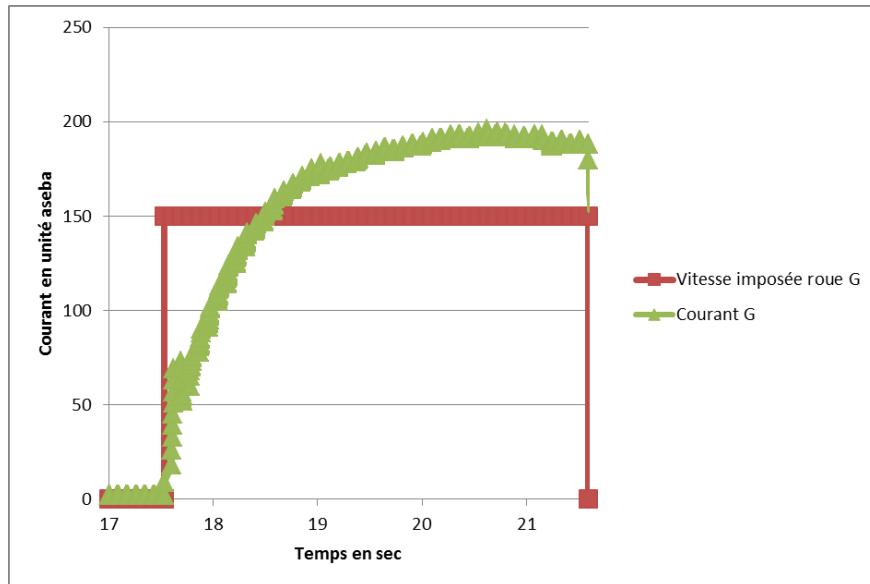


Figure 25: Mesure du courant (en unité aseba) en fonction du temps (en sec)

D'après la figure 25, on remarque que le courant augmente non linéairement et atteint une valeur maximale avec un certain retard. Similairement au profil de vitesse, le courant a une caractéristique exponentielle. Cela est dû au filtrage(présence de capacité) du signal de sortie.

#### Modélisation et choix du modèle

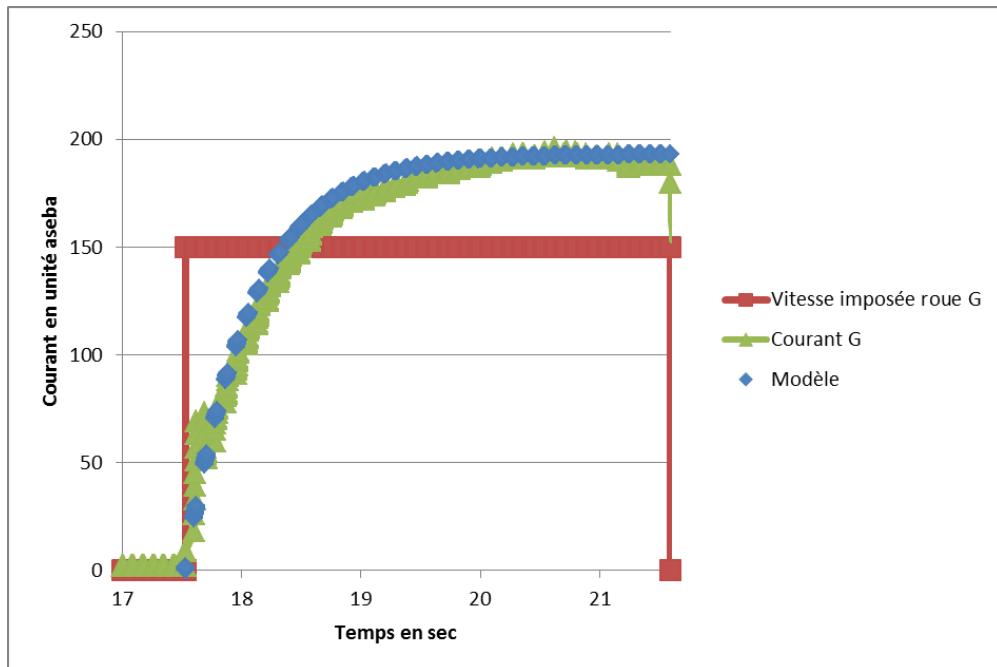


Figure 26: Mesure du courant (en unité aseba) en fonction du temps (en sec)

Pour établir le modèle du profil du courant, on a calculé le retard  $\tau$  du courant pour chaque robot et pour chaque vitesse de consigne entre 50 et 600 avec un pas de 50. Tous les tau calculés sont très similaires et sont indépendants de la vitesse (voir annexe D), et dont la moyenne totale vaut  $tau = 0.55s$ . Ainsi on peut établir l'équation suivante :

$$i(t) = i_{max} \cdot (1 - e^{-\frac{t}{\tau}}) \quad (8)$$

### 3.3.3 Caractéristique du bruit du courant i

#### Mesures et observations

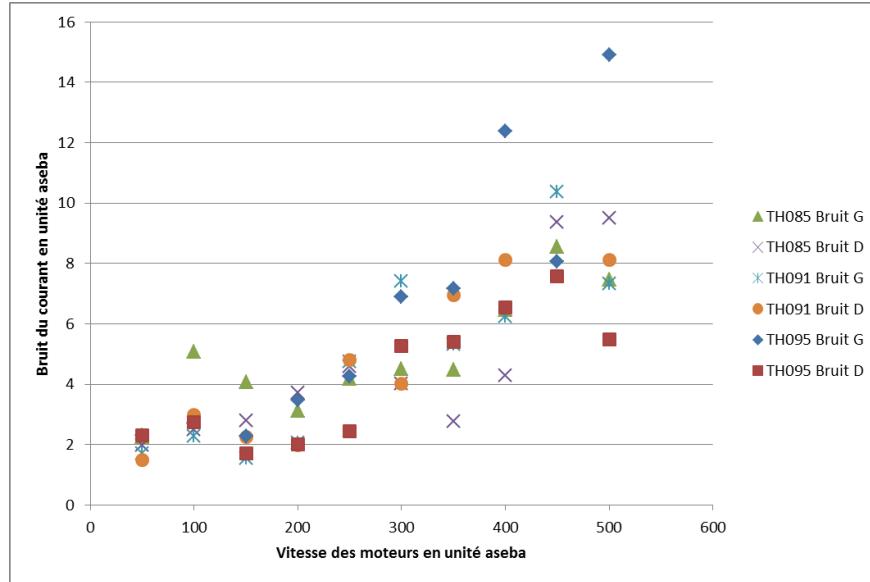


Figure 27: Mesure du bruit du courant (en ua) en fonction de la vitesse (en ua)

On constate que le bruit augmente plus la vitesse est grande, mais il reste cependant faible comparé à la moyenne du courant : pour une vitesse de 300ua, on a en moyenne un courant  $i = -306ua$  avec un bruit d'approximativement  $6ua$ .

#### Modélisation et choix du modèle

Similairement aux sections précédentes, j'ai défini un bruit ayant une caractéristique linéaire (voir fig. 28) et étant proportionnel à la vitesse des moteurs (en moyennant les droites de tendances du bruit des 6 moteurs). On obtient l'équation suivante en fonction de la vitesse  $v$  :

$$\sigma_{bruit} = 0.0162 \cdot v + 0.533 \quad (9)$$

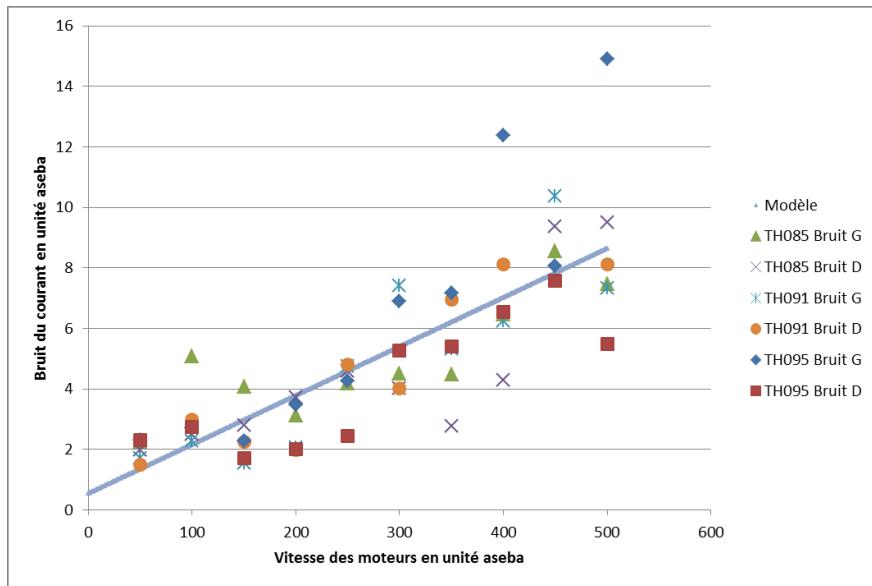


Figure 28: Comparaison entre le modèle et les mesures sur le bruit du courant (en ua) en fonction de la vitesse (en ua)

### 3.4 Proposition d'amélioration pour les moteurs

Concernant toutes les sous-sections précédentes (vitesse, PWM et courant), on pourrait améliorer le modèle simulé en proposant un paramètre de dispersion aléatoire pour les valeurs maximales, la pente des droites caractéristiques (ainsi que l'ordonnée à l'origine) ou encore les constantes de temps  $\tau$  selon une certaine plage de données. Cela permettrait de ne pas avoir 2 robots totalement identiques lors d'une simulation en incluant plusieurs.

Par exemple, on pourrait proposer des modèles simulés n'ayant pas les mêmes vitesses maximales pour les moteurs à gauche et à droite (phénomène observé chez les 3 robots de tests) pour être le plus proche du fonctionnement des robots réels.

## 4 Modélisation des capteurs verticaux

### 4.1 Caractéristique des capteurs

#### 4.1.1 Mesures et observations

Les capteurs verticaux permettent de mesurer les différents niveaux de gris de la surface sur laquelle il est posé étant donné que la distance entre le capteur et le sol reste constante. Mais ils permettent aussi de détecter les bords d'une table par exemple. Les variables affectées par les mesures dans aseba sont :

- *prox.ground.reflected* : qui mesure les ondes infrarouges que le capteur reçoit : la lumière ambiante ainsi que les ondes envoyées depuis le robot et reflété sur la surface.
- *prox.ground.ambiant* : qui mesure seulement la lumière ambiante lorsque l'émission d'onde IR depuis le robot est stoppée
- *prox.ground.delta* : qui effectue la différence entre les ondes reflétées et la lumière ambiante



Figure 29: Page 1 des niveaux de gris entre 100% et 30%

Les mesures se sont effectuées durant 3 secondes sur des niveaux de gris (voir fig. 29) entre 0% et 100% avec un pas de 10%. Le premier problème rencontré concernait l'impression car selon les feuilles et l'imprimante, les niveaux de gris ne sont pas du tout interprétés par les capteurs verticaux du robot de la même manière. Ainsi on a effectué les mesures sur les feuilles ayant été imprimées sur 4 imprimantes différentes (voir fig. 30) : une imprimante noir et blanc, une imprimante couleur EPFL, une imprimante couleur classique tout public (ou imprimante "maison") et une imprimante couleur du labo LSRO.



Figure 30: Différence du ressorti des 4 premiers niveaux de gris (entre 100% et 70%) selon les imprimantes respectivement citées précédemment

Ainsi pour chaque différentes impressions et pour chaque robot, on a mesuré la moyenne du signal sur 3 secondes pour chaque niveau de gris en s'attendant à obtenir un résultat proportionnel entre les mesures et les niveaux de gris théoriques. La figure 31 met en évidence la différence de mesures faites avec le robot TH091 pour chacune de ces impressions (voir annexe H pour les robots TH085 et TH095).

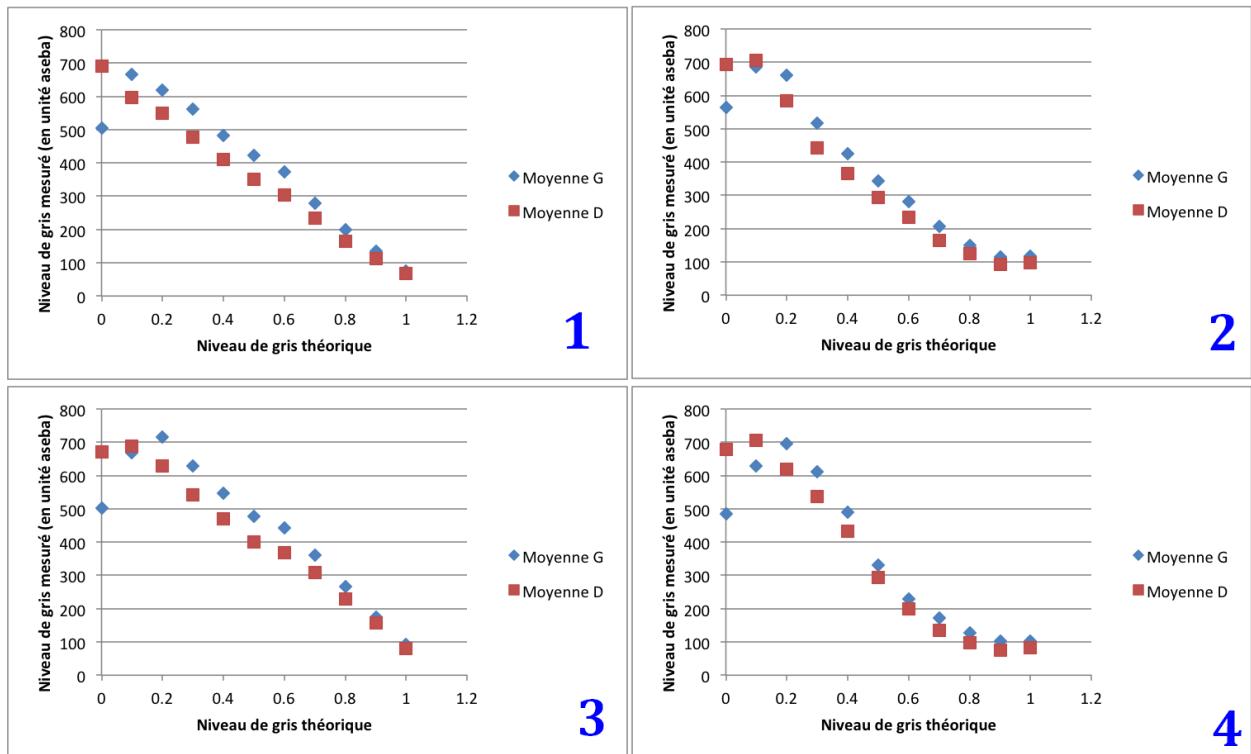


Figure 31: Différence entre les valeurs mesurées par les capteurs en unité aseba en fonction du niveau de gris théorique (entre 0 et 1) pour les 4 feuilles avec le robot TH091 : 1. imprimante noir et blanc, 2. imprimante couleur EPFL, 3. imprimante couleur "maison", 4. imprimante couleur du labo LSRO

Pour les impressions 1 et 3, les niveaux de gris mesurés augmentent effectivement proportionnellement en fonction du niveau de gris théorique. Concernant, les impressions 2 et 4 qui sont très similaires, on remarque que les niveaux de gris entre 100% et 80% ne présentent pas énormément de différences dans les mesures : le niveau de gris 90% est même plus sombre que celui à 100% (ce qui peut aussi s'observer dans la réalité sur la figure 30).

Ainsi pour le choix du modèle les impressions 2 et 4 ont été écartés du fait de leur non linéarité. Les mesures effectuées sur l'impression 1 donnent de meilleurs résultats que pour l'impression 3, c'est pourquoi le modèle simulé sera basé sur les résultats obtenus grâce aux mesures effectuées sur les feuilles de l'imprimante 1.

On peut aussi remarquer que les capteurs saturent lorsqu'on essaie de capturer des niveaux de gris faible (entre 0% et 20%) et que la lumière ambiante est trop importante. Les valeurs maximales que le capteur renvoie dans aseba se situe environ à 1000ua. Le capteur 1 (à droite) du robot TH085 est un peu défaillant mais illustre bien ce problème de saturation pour les gris clairs ou blanc (voir fig. 32).

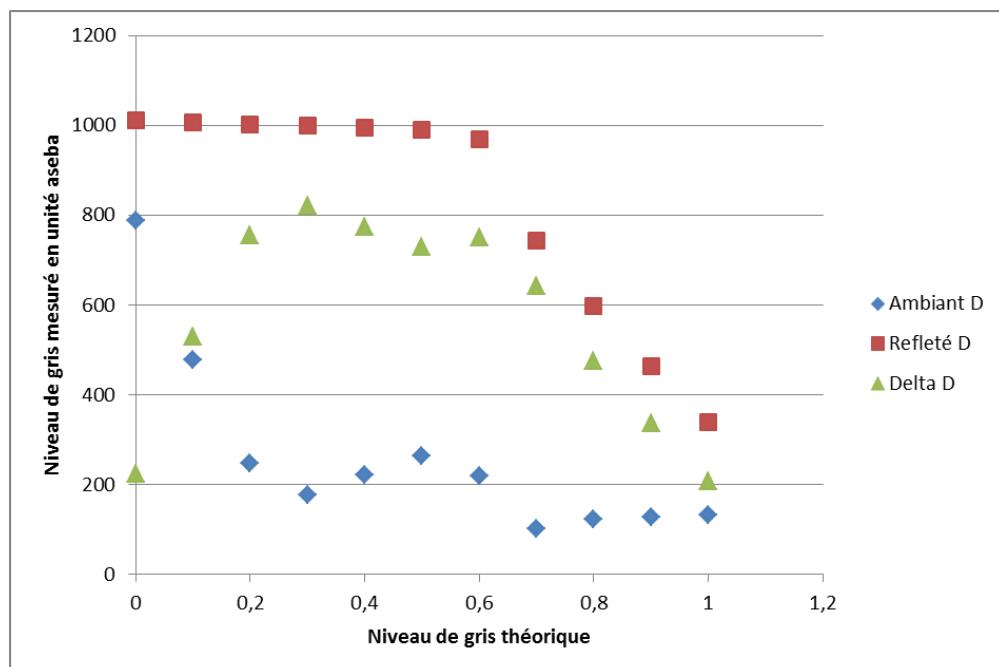


Figure 32: Mesure des 3 variables lumière ambiante, lumière réfléchie et le delta (entre la lumière réfléchie et la lumière ambiante) exprimés en unité aseba en fonction du niveau de gris théorique

D'après la figure 32, le capteur défaillant sature très vite (dès les niveaux de gris inférieurs à 60%), mais on observe du coup que la différence delta est faussée lorsque la lumière réfléchie capturée est saturée car plus le bruit devient important, plus la différence diminuera : dans notre cas le capteur ne fait pas la différence entre le blanc et le noir. Ce phénomène est similaire pour les autres capteurs mais à des niveaux de gris plus faibles (inférieurs à 20%).

On a alors effectué les mesures sans présence de lumière pour avoir une lumière ambiante nulle (renvoyée dans aseba) et ainsi ne capturer que la lumière reflétée sur les feuilles pour les 3 robots (voir fig. 33).

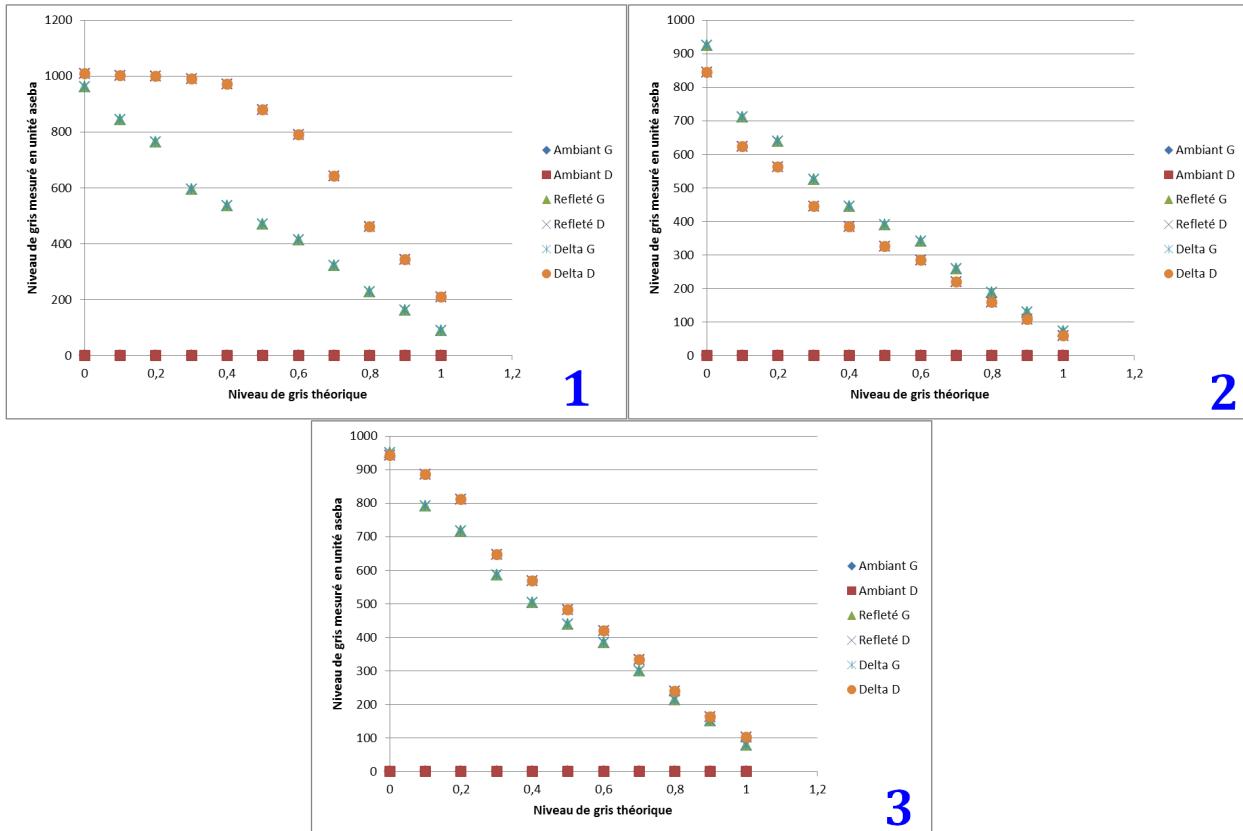


Figure 33: Mesure dans l'obscurité des 3 variables des 2 capteurs (lumière ambiante, lumière reflétée et la différence des deux exprimés en unité aseba) en fonction du niveau de gris théorique pour le robot TH085 (1), TH091 (2) et TH095 (3)

D'après la figure ci-dessus, la lumière ambiante valant effectivement 0, la lumière reflétée est égale au delta et mis à part le capteur défaillant (capteur de droite du robot TH085), la relation linéaire entre les mesures et les niveaux de gris théoriques est très marquée et elle sera la base de l'établissement du choix du modèle pour le Thymio II simulé.

#### 4.1.2 Modélisation et choix du modèle

L'établissement du modèle de la caractéristique du niveau de gris est une fonction linéaire valant la moyenne de 5 des 6 droites de tendances effectuée à partir des mesures des moyennes du niveau de gris. On calcule ainsi une droite d'équation :

$$g_v = -806.93 \cdot g_t + 853.24 \quad (10)$$

avec  $g_v$  la valeur du niveau de gris en unité aseba et  $g_t$  la valeur du niveau de gris de l'image lue entre 0 et 1.

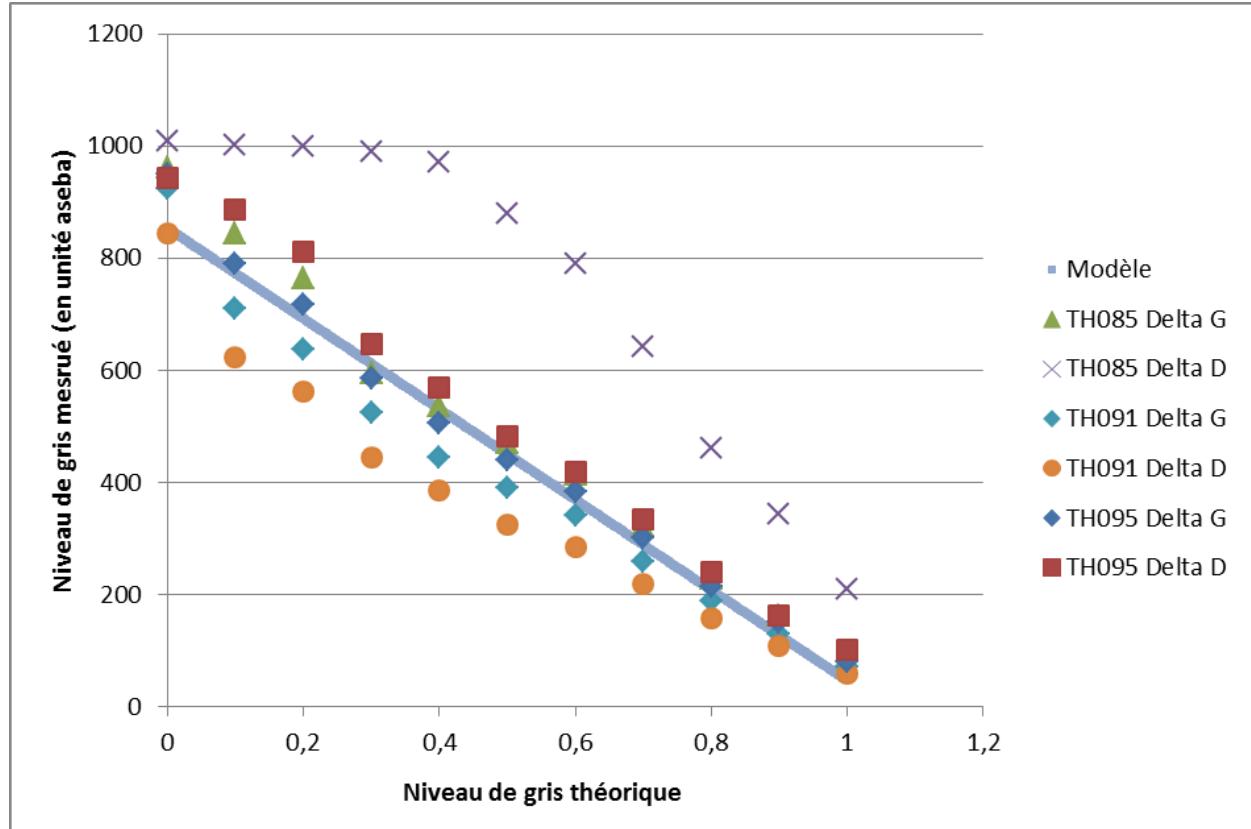


Figure 34: Modèle du niveau de gris (en unité aseba) en fonction du niveau de gris théorique

Concernant l'implémentation du modèle sous enki, on fait que choix que la fonction simulant les capteurs horizontaux renvoie la valeurs d'une pixel de la texture au sol. En réalité, le robot réel capte une zone dont il fait la moyenne, c'est pourquoi on a appliqué un filtre gaussien de taille 9x9 sur la texture afin que le modèle du capteur simulé soit le plus proche de la réalité.

## 4.2 Caractéristique du bruit des capteurs

### 4.2.1 Mesures et observations

Les conditions de mesures pour le bruit sont les mêmes que pour les calculs de la moyenne, sauf qu'on calcule l'écart type. On a effectué les mesures sur chaque robot et chaque feuille, mais celle retenue pour le modèle sont celles effectuées sur les impressions 1 (voir fig. 29 et 30) dans l'obscurité (pour ne pas prendre en compte le bruit dû à une source de lumière externe).

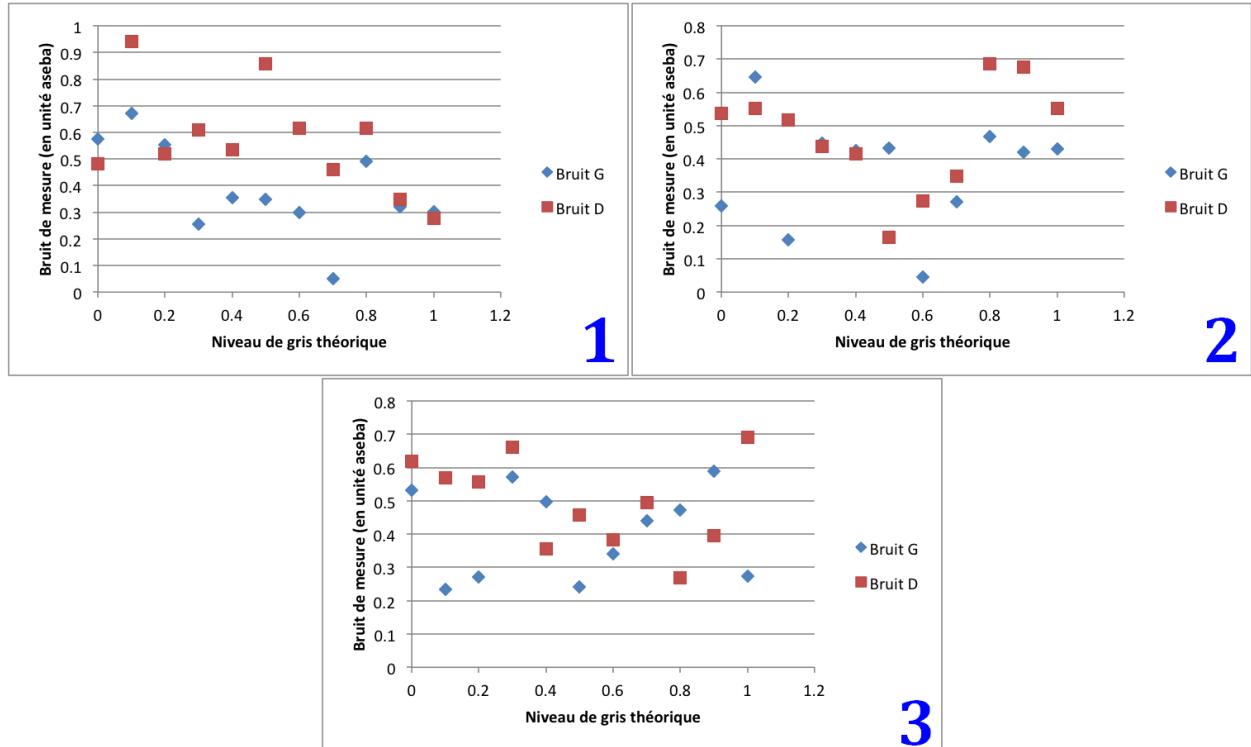


Figure 35: Bruit des capteurs au sol (en unité aseba) en fonction du niveau de gris théorique dans l'obscurité. Robots : TH085 (1), TH091 (2) et TH095 (3)

Sur la figure 35, on remarque que le bruit n'est pas proportionnel au niveau de gris théorique et il est aussi très faible comparé aux moyennes : il est toujours inférieur à 1ua comparé à des moyennes valant entre 850ua et 50ua.

### 4.2.2 Modélisation et choix du modèle

Le bruit de mesure concernant les capteurs au sol est donc déterminé par une constante égale à la moyenne des toutes les valeurs du bruit mesurés et valant  $\sigma = 0.45ua$ .

## 5 Modélisation des LEDs

### 5.1 Mesures et observations

Il n'est pas possible de mesurer une variable liée au LEDs dans Aseba Studio. Le modèle s'est seulement basé sur des observations.

Tout d'abord le robot contient 34 paramètres de LEDs dont les valeurs sont attribuées entre 0 et 32 selon le code aseba par appel de fonction; 2 LEDs ne sont pas contrôlables par l'utilisateur : les LEDs indiquant le niveau de batterie et la LEDs d'indication de connexion USB.

Par exemple le code suivant activera 2 des LEDs des boutons avec une intensité définie à 32 et 5 respectivement (voir fig. 36) :

```
// call leds.buttons(32, 0, 5, 0)
```



Figure 36: Robot Thymio II dont 2 LEDs des bouttons sont allumés avec différentes intensités

Ces paramètres modifiables sont :

- 8 paramètres pour les 8 LEDs du cercle entourant les boutons (intensité)
- 4 paramètres correspondant aux 4 LEDs situés en dessous des boutons
- 8 paramètres pour les LEDs situés à côté des capteurs horizontaux
- 2 paramètres pour les LEDs situés à côté des capteurs verticaux
- 1 paramètres pour la LED indicatrice de l'activation d'un son
- 1 paramètre pour la LED indicatrice de communication IR avec télécommande
- 2 paramètres pour définir le niveau de bleu et de rouge des 2 LEDs pour l'indicateur de température
- 3 paramètres pour définir le niveau de bleu, vert et rouge des LEDs sur le dessus du robot
- 3 paramètres pour définir le niveau de bleu, vert et rouge des LEDs sur le coté gauche du robot
- 3 paramètres pour définir le niveau de bleu, vert et rouge des LEDs sur le coté droit du robot

## 5.2 Modélisation et choix du modèle

Ainsi le modèle ne peut être représenté que sous forme de texture de LEDs qui s'ajoute à celle du robot simulé en 3D, pour simuler si elle est allumée et avec quelle intensité. L'équivalent de l'intensité dans le programme est défini par la couche alpha (c'est à dire la transparence) de la texture de la LED. La couche alpha prend une valeur entre 0 et 255 et qui équivalent au paramètre (entre 0 à 32) défini dans l'appel de fonction des LEDs sous aseba. Un problème est apparu lors des tests en codant le programme suivant :

```
|| call leds.prox.h(0,1,5,15,25,32,0,0)
```

Les LEDs 0 à 5 des capteurs horizontaux ont respectivement pris les valeurs 0, 1, 5, 15, 25 et 32 alors que les 2 LEDs arrières (correspondant au capteurs horizontaux à l'arrière du robot) sont restées désactivées. On observe les affichages suivants :

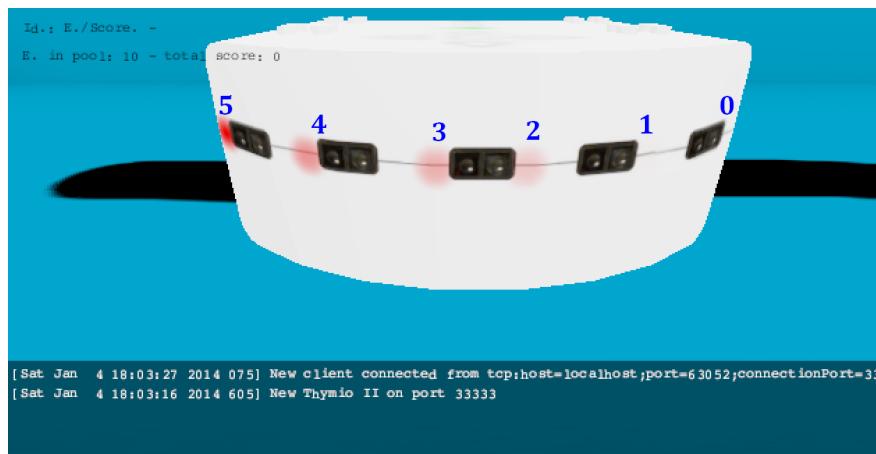


Figure 37: Première tentative d'affichage des textures des LEDs



Figure 38: Affichage des LEDs selon le code défini précédemment

La LED 1 du modèle simulé n'est pas du tout visible alors qu'elle l'est dans le modèle réel même si l'intensité est fixée à 1 dans le code aseba.

Le problème a été réglé en fixant une valeur minimum à la couche alpha valant ainsi entre 80 et 255 pour des intensités fixées entre 0 et 32 dans le code aseba (32 équivalent à un alpha = 255 et 1 équivalent à alpha = 80).

## 6 Implémentation du code dans enki et Aseba Playground

### 6.1 Première implémentation d'un Thymio dans Enki et Aseba

#### 6.1.1 Lien entre Aseba et Enki pour le Thymio II

La première étape a été de créer le lien entre une nouvelle classe dans Enki et une nouvelle classe dans Aseba Playground du Thymio II. La classe AsebaThymio2 (voir Annexe F) hérite de la classe Thymio2 (voir Annexe F) et contient essentiellement les variables, la liste des événements et les fonctions natives propres au Thymio.

Le fichier thymio2.cpp dans le dossier playground contient les fonctions natives, leurs descriptions (reprises du firmware), une fonction de mise à jour des variables du robot faisant le lien avec le modèle simulé (sous Enki afin qu'elle puisse être lu et paramétrable dans Aseba Studio) et des fonctions responsables des évènements.

Le fichier thymio2.cpp dans le dossier robot\thymio2 de la bibliothèque Enki contient les variables des capteurs horizontaux et verticaux et les paramètres des LEDs. La classe Thymio2 définie dans le header hérite quant à elle de la classe DifferentialWheeledThymio2 qui gère le calcul de la vitesse, du PWM et du courant des moteurs.

#### 6.1.2 Définition de la forme physique du robot

Le design de base du Thymio II pouvant être visualisable sous Playground était un cylindre ayant pour rayon la distance entre le centre du support à crayon et l'avant du robot (en arc de cercle) : c'est à dire 8cm. Des fonctions contenues dans le fichier PhysicalEngine.cpp (gérant la physique des objets dans le monde 2D simulé par Enki) ont permis de construire un objet physique ayant une forme personnalisée s'approchant au maximum du Thymio II en définissant point par point les limites à définir (en rouge sur la fig. 39).

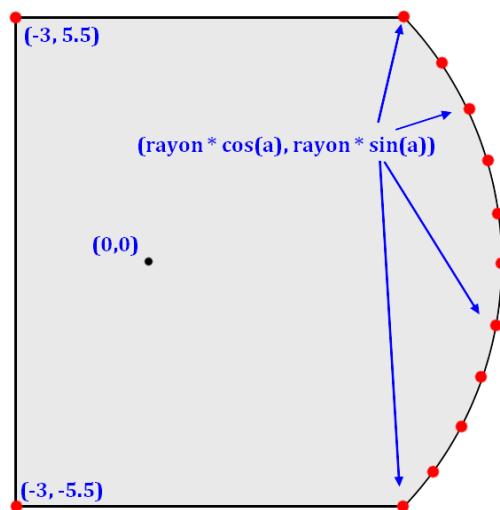


Figure 39: Position des points définissants la forme physique du Thymio II par rapport au centre du support crayon

Le code permettant d'établir cette objet est définit ci-dessous :

```
Enki::Polygone thymio2_polygone;
    const double amount = 10.0;
    const double radius = 8.0;
    const double height = 5.0;
    const double angle1 = asin(5.5/8.0);
    const double angle2 = atan(5.5/3.0);
    const double distance = sqrt(3.0*3.0+5.5*5.5);

    // Defining physical shape of the Thymio
    for (double a = -angle1; a < angle1+0.01; a += 2*angle1/amount)
        thymio2_polygone.push_back(Enki::Point(radius * cos(a), radius * sin(
            a)));

    thymio2_polygone.push_back(Enki::Point(distance * cos(M_PI - angle2),
        distance * sin(M_PI - angle2)));
    thymio2_polygone.push_back(Enki::Point(distance * cos(M_PI - angle2),
        distance * sin(M_PI + angle2)));
    Enki::PhysicalObject::Hull hull(Enki::PhysicalObject::Part(
        thymio2_polygone, height));

    setThymio2Hull(hull, 1);
```

la fonction *setThymio2Hull* permet quand à elle de finaliser l'objet physique en calculant sa masse et son moment d'inertie. Mais contrairement la fonction *setHull*, elle ne recalcule pas son centre de masse. Ainsi le point (0, 0) du robot est considéré comme le centre de masse comme indiqué sur la figure 39 (c'est à dire au centre du support crayon). Il est aussi utilisé comme point de pivot pour les vitesses augulaires.

**Remarque :** le centre du support crayon se situe exactement au milieu des deux roues et est effectivement le point de pivot du robot.

Sa forme physique n'est cependant pas visualisable si on affiche le modèle 3D réalisé sous Blender. Il est seulement défini pour les calculs d'interaction physique (collisions, mouvements, moments d'inertie, etc.) dans l'environnement 2D simulé.

### 6.1.3 Design 3D sous Blender

Le design a été récupéré depuis un modèle déjà existant créé sur PTC Pro/Engineer Wild-Fire 4.0. Afin de pouvoir être affiché grâce à la bibliothèque OpenGL, le fichier a été enregistré dans le format .obj afin d'être transcrit en fichier .cpp à l'aide d'un petit script ("obj2opengl.py" réalisé par Heiko Behrens).

Le premier modèle 3D comportait plus de 27000 faces et plus de 13000 sommets. De plus, énormément de triangles dégénérés perturbait l'affichage. Le modèle a donc été simplifié sous Blender afin d'optimiser le nombre de faces et de sommets ainsi que de supprimer les triangles dégénérés.

Le modèle final comporte près de 800 faces et 500 sommets, ce qui rend le fichier beaucoup plus léger et les calculs plus rapide pour l'affichage (voir fig. 40). Le même travail a été effectué pour les roues du Thymio II qui sont traitées séparément afin de pouvoir pivoter lors du fonctionnement des moteurs (simulant ainsi la rotation des roues).

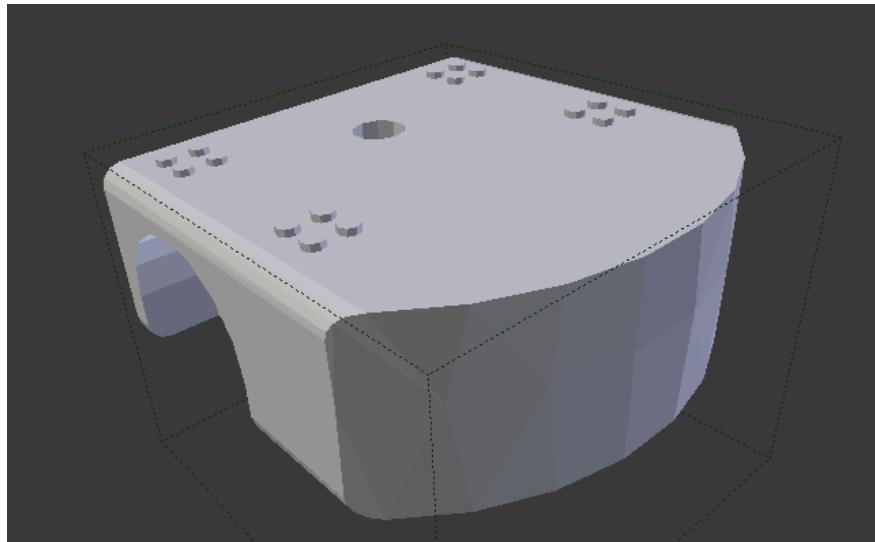


Figure 40: Design simplifié du corps du Thymio II sous Blender

L'UV-Map de l'objet sous Blender a permis l'établissement d'une texture dessinée à l'aide d'un logiciel de retouche d'image, en dessinant manuellement ou encore en ajoutant des photos de certains composants (les capteurs, l'entrée USB et le lecteur de carte SD). On a ainsi établi la texture qui est appliquée au modèle 3D du robot (voir fig. 41).



Figure 41: Affichage du Thymio II sous Aseba Playground

## 6.2 Les événements

Les événements définis dans la section 1.4 ont pu être implémentés à l'aide d'un timer (récupéré grâce à la bibliothèque QTimer). On a défini une nouvelle classe *ThymioTimer* (voir Annexe F) permettant d'initialiser un Timer de fréquence 1000Hz appelant une fonction activant les flags des événements. La fonction est définie comme suit :

```
|| void AsebaThymio2::timer_update()
|| {
||     if(counter % 10 == 0)
```

```

        SET_EVENT(EVENT_MOTOR);
    if(counter % 50 == 0)
        SET_EVENT(EVENT_BUTTONS);
    if(counter % 100 == 0)
        SET_EVENT(EVENT_PROX);
    if(counter % 62 == 0)
        SET_EVENT(EVENT_ACC);
    if(counter % 1000 == 0)
        SET_EVENT(EVENT_TEMPERATURE);
    if (variables.timerperiod[0] > 0) {
        if(counter % variables.timerperiod[0] == 0)
            SET_EVENT(EVENT_TIMER0);
    }
    if (variables.timerperiod[1] > 0) {
        if(counter % variables.timerperiod[1] == 0)
            SET_EVENT(EVENT_TIMER1);
    }
    int maxcount; // !\ Warning : works for 32bit and 64bit only
    if(variables.timerperiod[0] > 0 && variables.timerperiod[1] > 0)
        maxcount = 62 * 1000 * variables.timerperiod[0] * variables.
            timerperiod[1];
    else if(variables.timerperiod[0] > 0 && variables.timerperiod[1] == 0)
        maxcount = 62 * 1000 * variables.timerperiod[0];
    else if(variables.timerperiod[0] == 0 && variables.timerperiod[1] > 0)
        maxcount = 62 * 1000 * variables.timerperiod[1];
    else
        maxcount = 62 * 1000;

    counter++;
    if(counter == maxcount)
        counter = 0; //re-initialize the counter
}

```

Le compteur "counter" permet de déterminer quand il faut activer les flags à l'aide des conditions et s'adapte en fonction des périodes définies pour les évènements *timer0* et *timer1* afin que afin que le compteur se réinitialise sur un multiple des périodes et éviter qu'un flag d'événement ne soit pas mis à 1 au bout d'un temps supérieur la période prédéfinie.

### 6.3 Les capteurs horizontaux

Le modèle est définie selon l'équation (1) dans la section 2.1.2 et son implémentation avait déjà été réalisée pour l'E-puck par la classe *IRSensor*. Cependant j'ai adapté 4 lignes code dans le fichier *IRSensor.cpp* afin de supprimer le bruit sur la réponse du capteur si la valeur vaut 0 :

```

#define MIN_TOL 0.001

if(finalValue > MIN_TOL)
    finalValue = std::max(0., std::min(m, gaussianRand(finalValue, noiseSd)));
else
    finalValue = std::max(0., std::min(m, gaussianRand(finalValue, 0)));

```

La création d'un nouveau capteur est définie par le code suivant :

```

IRSensor(Robot *owner, Vector pos, double height, double orientation, double
    range, double m, double x0, double c, double noiseSd = 0.);

```

avec *\*owner* un pointeur sur le robot actuel, *pos* sa position en cm dans l'univers 2D simulé, *height* la hauteur du capteur par rapport au sol, *orientation* son orientation par rapport à l'axe x du robot, *range* sa portée, *m* la valeur maximale renvoyée, *x0* la distance à laquelle on obtient la valeur maximale *m*, *c* un paramètre de calibration et *noiseSd* le bruit fixé sur le capteur.

**Rappel :** Ces paramètres ont été déterminés dans la section 2 et ont les valeurs suivantes :

- $x_0 = 0.03\text{cm}$
- $m = 4505\text{ua}$
- $c = 73.29$
- $\text{range} = 14\text{cm}$

La figure 7 situe les capteurs horizontaux sur le robot.

On a calculé la position des capteurs par rapport à son centre (le centre du support crayon), leur hauteur ainsi que leur orientation et on a initialisé les 7 capteurs selon le code suivant :

```
infraredSensor0(this, Vector(6.5, 4.9), 3.5, 0.66491, 14, 4505, 0.03, 73.29,
                2.87),
infraredSensor1(this, Vector(7.6, 2.5), 3.5, 0.31780, 14, 4505, 0.03, 73.29,
                2.87),
infraredSensor2(this, Vector(8.0, 0.0), 3.5, 0, 14, 4505, 0.03, 73.29, 2.87),
infraredSensor3(this, Vector(7.6, 2.5), 3.5, 0.31780, 14, 4505, 0.03, 73.29,
                2.87),
infraredSensor4(this, Vector(6.5, -4.9), 3.5, 0.66491, 14, 4505, 0.03, 73.29,
                2.87),
infraredSensor5(this, Vector(-3.0, 3.0), 3.5, -M_PI, 14, 4505, 0.03, 73.29,
                2.87),
infraredSensor6(this, Vector(-3.0, -3.0), 3.5, -M_PI, 14, 4505, 0.03, 73.29,
                2.87)
```

On a programmé un robot réel (TH095) et un robot simulé afin de comparer les valeurs que renvoie le capteur 2 (situé au devant du robot) lorsque le robot s'éloigne avec une vitesse de consigne de -50 :

```
motor.left.target = -50
motor.right.target = -50

onevent prox
emit capteur prox.horizontal[2]
```

Sur la figure 42, on peut observer que les réponses du robot simulé et du robot réel sont très similaires. Les valeurs déviante sur le modèle pour les distances limites maximales (aux alentours de 18 secondes sur la figure) sont dues que la différence de vitesse entre les 2 roues étaient non négligeable et que le robot a légèrement dévié de sa trajectoire lors des mesures. La différence de réponse entre la roue gauche et la roue droite pour le robot simulé n'a pas été implémentée.

On peut donc conclure que le modèle simulé des capteurs horizontaux est relativement fidèle au comportement d'un robot réel.

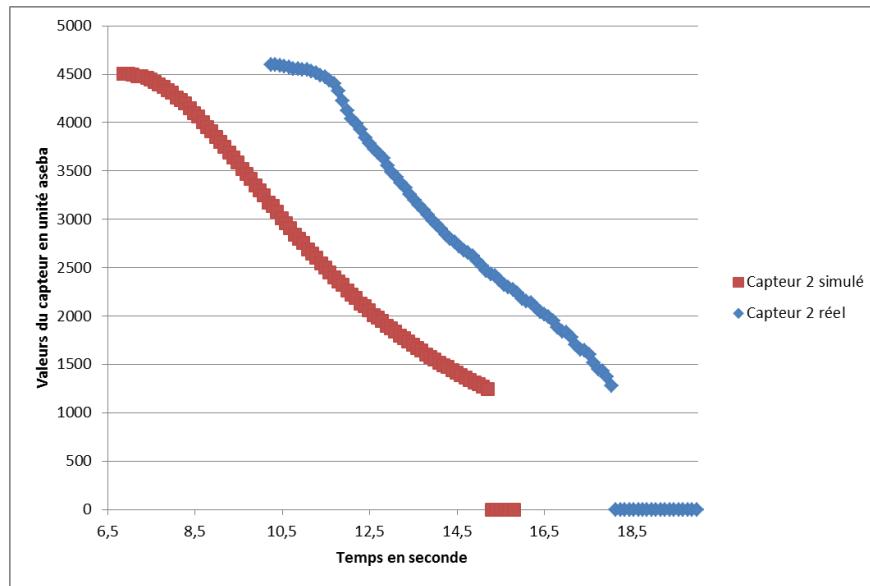


Figure 42: Comparaison entre la réponse d'un robot réel et d'un robot simulé

## 6.4 Les moteurs

Les modèles liés au moteur concernent la vitesse, le PWM et le courant. Ils ont été établis par rapport à leurs profils, leurs valeurs maximales (en moyenne) et leurs bruits.

Une nouvelle classe a été créée pour le fonctionnement et la simulation des moteurs pour le Thymio II et se dénomme *DifferentialWheeledThymio2* (voir Annexe F) et contient les variables donnant la vitesse, le PWM et le courant pour chaque moteur. Il contient aussi les variables de la vitesse angulaire et la vitesse tangeante du robot.

### Rappel

La vitesse maximale a été fixée à 500. Son profil suit une courbe exponentielle (similaire au circuit RC) et est définie selon l'équation suivante :

$$v(t) = v_{max} \cdot (1 - e^{-\frac{t}{\tau}}) \quad (11)$$

La constante de temps choisie pour le modèle vaut  $\tau = 0.025$ .

La caractéristique du bruit est définie par une fonction du 1er degré et est définie par l'équation suivante :

$$\sigma_{bruit} = 0.053v + 0.85 \quad (12)$$

avec  $v$  la vitesse des moteurs.

Le PWM maximal est fixé à 800, et son profil est très peu visible à cause du bruit. Cependant, il est proportionnel à la vitesse, et suit le même profil que cette dernière. La moyenne de ses valeurs augmente en fonction de la vitesse selon l'équation suivante :

$$PWM = -1.2806 \cdot v - sgn(v) \cdot 142.14 \quad (13)$$

Et le bruit augmente aussi en fonction de la vitesse selon l'équation suivante :

$$\sigma_{bruitPWM} = 0.3629 \cdot v + 7.01 \quad (14)$$

Pour terminer, le courant maximum est proportionnel à la vitesse des moteurs :

$$i(v) = -0.5572 \cdot v - \text{sgn}(v) \cdot 139.32 \quad (15)$$

Son profil, tout comme le profil de vitesse, suit une courbe exponentielle similaire à celle d'un circuit RC :

$$i(t) = i_{max} \cdot (1 - e^{-\frac{t}{\tau}}) \quad (16)$$

Sa caractéristique du bruit est une fonction du premier degré, proportionnelle à la vitesse et définie comme suit :

$$\sigma_{bruit} = 0.0162 \cdot v + 0.533 \quad (17)$$

## Implémentation

Concernant le profil de la vitesse et du courant, la formule a été un peu adaptée dans le code, car il faut prendre en compte la limite inférieure (c'est à dire la valeur minimale du profil, ce n'est pas nécessairement 0) et la limite supérieure (la valeur maximale à laquelle le signal tend en régime permanent). La figure 43 illustre cette idée.

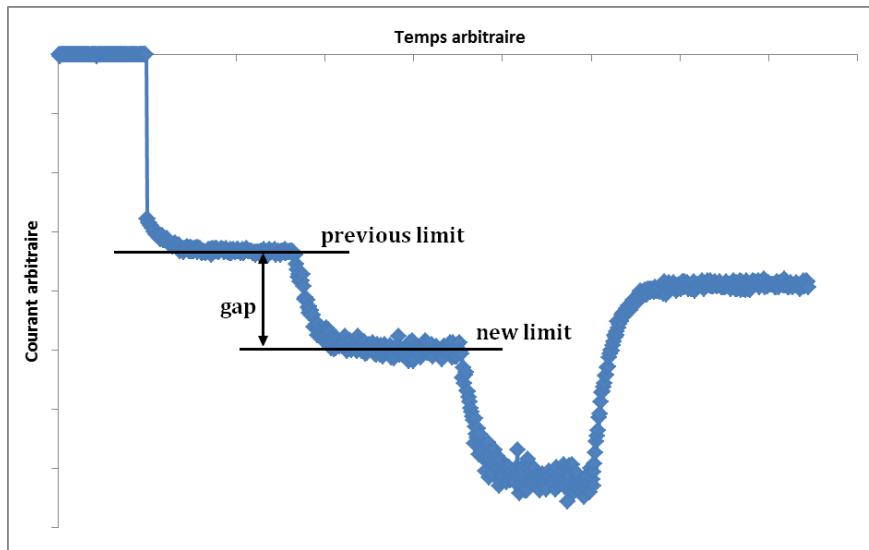


Figure 43: Profil de courant arbitraire en fonction du temps

La fonction suivante :

```
// void controlStep(double dt);
```

fait partie de la classe DifferentialWheeledThymio2 et a pour argument de delta de temps  $dt$  entre deux appels de cette fonction.

```
// Detect if there is a change of target speed and gives limit to speed and
// current
// Limits for left wheel
if(fabs(previousLeftSpeedTarget - leftSpeedTarget) > TOLERANCE) {
    n = 0;
}

// Limit Definition for speed profile
```

```

leftnewlimitSpeed = leftSpeedTarget;
leftpreviouslimitSpeed = previousleftSpeedTarget;

// Limit definition for current profile
leftnewlimitCurrent = CURRENT_FACTOR * (leftnewlimitSpeed * MAXSPEED_ASEBA /
    MAXSPEED_CM) + sgn(leftnewlimitSpeed)*CURRENT_BASE;
leftpreviouslimitCurrent = CURRENT_FACTOR * (leftpreviouslimitSpeed *
    MAXSPEED_ASEBA / MAXSPEED_CM) + sgn(leftnewlimitSpeed)*CURRENT_BASE;
}

```

On définit certaines variables utiles au calcul :

```

double timeleft = n*dt;
double timeright = m*dt;

double noiseSpeed = 0;
double noisePWM = 0;
double noiseCurrenti = 0;
double gap;

double target;

```

et on implémente tous les modèles selon le code suivant :

```

// Calculating speed
gap = leftnewlimitSpeed - leftpreviouslimitSpeed;
target = clamp(leftSpeedTarget, -maxSpeed, maxSpeed);
leftSpeed = (target - gap * exp(-timeleft/TAU_SPEED));
if (leftSpeed > TOLERANCE)
    noiseSpeed = leftSpeed*noiseFactor + noiseBase;
else
    noiseSpeed = 0;

// Calculating Current
gap = leftnewlimitCurrent - leftpreviouslimitCurrent;
target = CURRENT_FACTOR * (target * MAXSPEED_ASEBA / MAXSPEED_CM) + sgn(target)
    * CURRENT_BASE;
leftCurrenti = (target - gap * exp(-timeleft/TAU_CURRENT));
noiseCurrenti = CURRENT_NOISEFACTOR * (leftSpeed * MAXSPEED_ASEBA /
    MAXSPEED_CM) + CURRENT_NOISEBASE;

// Calculating PWM
if (leftSpeed > TOLERANCE) {
    leftPWM = PWM_FACTOR * (leftSpeed * MAXSPEED_ASEBA / MAXSPEED_CM) + sgn(
        leftSpeed) * PWM_BASE;
    noisePWM = PWM_NOISEFACTOR * (leftSpeed * MAXSPEED_ASEBA / MAXSPEED_CM) +
        PWM_NOISEBASE;
}
else
{
    leftPWM = 0;
    noisePWM = 0;
}

// Adding noise
leftSpeed = gaussianRand(leftSpeed, noiseSpeed);
leftPWM = gaussianRand(leftPWM, noisePWM);
if(abs(leftPWM) > 801)
    leftPWM = sgn(leftPWM)*801;
leftCurrenti = gaussianRand(leftCurrenti, noiseCurrenti);

```

**Remarques :** la même démarche est appliquée pour la roue droite.

Les constantes présentes dans le code sont définies au début du code et représentent les paramètres définissants les fonctions du premier degré ou les fonctions exponentielles à régime transitoire et régime permanent :

```
#define MAXSPEED_ASEBA 500      // Maximum speed (aseba values)
#define MAXSPEED_CM   16.6       // Maximum speed (cm/s)
#define PWM_FACTOR    -1.2806
#define PWM_BASE     -142.14
#define CURRENT_FACTOR -0.5572
#define CURRENT_BASE   -139.32
#define TOLERANCE     0.01
#define TAU_SPEED      0.025
#define TAU_CURRENT    0.55
#define CURRENT_NOISEFACTOR 0.0162
#define CURRENT_NOISEBASE 0.533
#define PWM_NOISEFACTOR 0.3629
#define PWM_NOISEBASE 7.01
```

On a ensuite effectué le code aseba suivant sur le modèle simulé et sur un robot réel (TH095) :

```
var mesure_moteur[8] = 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
var v0 = 0
var v1 = 0

motor.left.target = 0
motor.right.target = 0
timer.period[0] = 2000
timer.period[1] = 4000

onevent motor
mesure_moteur = [ motor.left.target, motor.right.target, motor.left.speed,
                  motor.right.speed, motor.left.pwm, motor.right.pwm, _imot[0], _imot[1] ]
emit moteur mesure_moteur

onevent timer0
v0 = v0 + 50
v1 = v1 + 50
motor.left.target = v0
motor.right.target = v1

onevent timer1
motor.left.target = 0
motor.right.target = 0
```

Toutes les 4 secondes, la vitesse de consigne du robot augmente de 50ua pendant 2 secondes et elle reprend la valeur de 0. On obtient ainsi le graphique suivant pour le modèle réel :

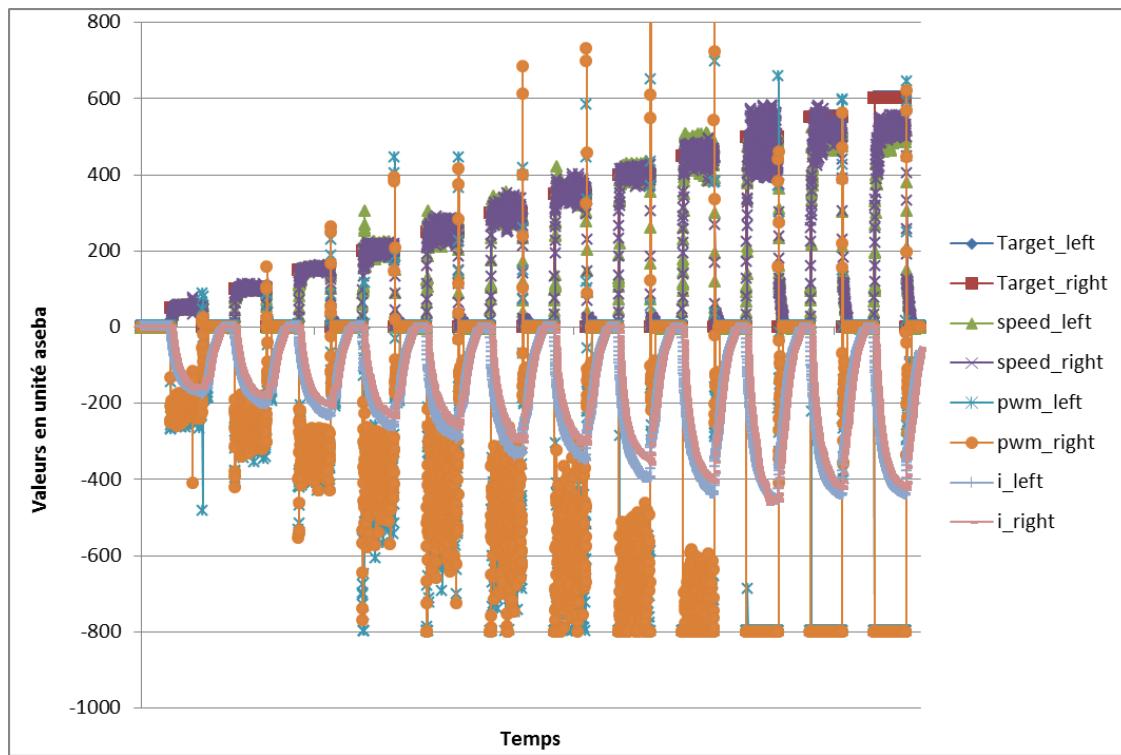


Figure 44: Profil de vitesse, du PWM et du courant pour le robot réel en fonction du temps

Pour le modèle simulé, on obtient le graphique suivant :

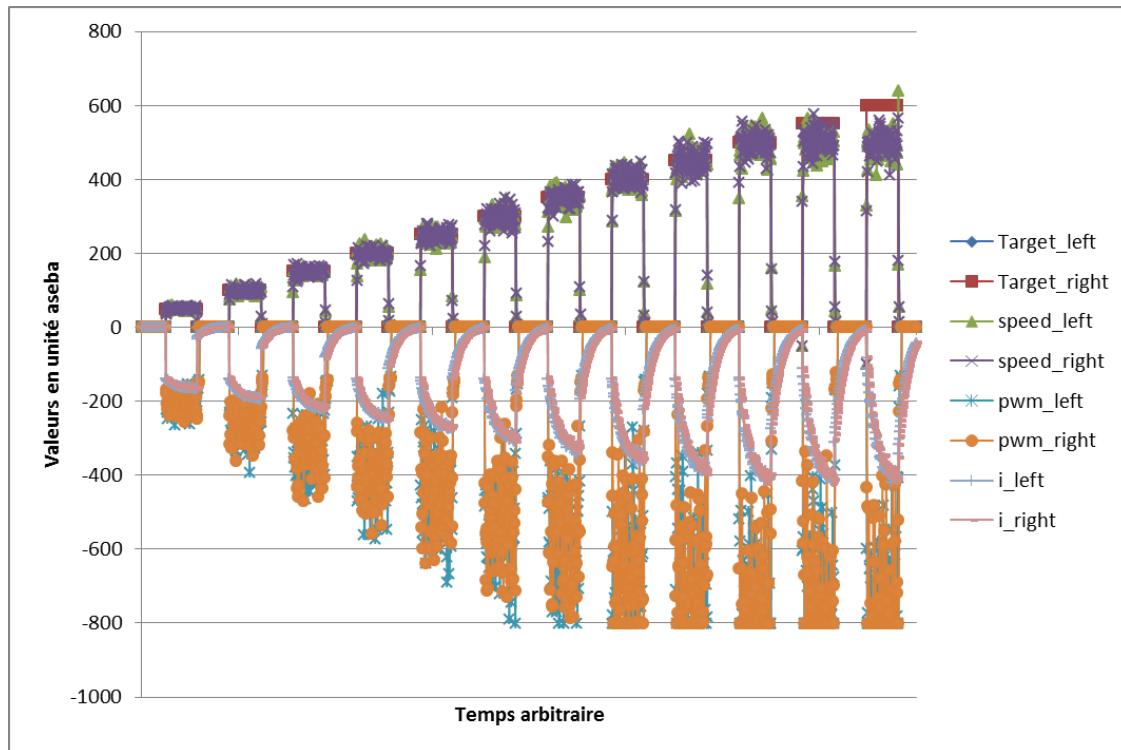


Figure 45: Profil de vitesse, du PWM et du courant pour le robot simulé en fonction du temps

On constate que le modèle reste fidèle à la réalité à part pour le PWM dont la pente moyenne

semble similaire mais dont le bruit est beaucoup plus important pour des vitesses supérieures à 400 sur le modèle simulé que sur le robot réel.

Globalement, on observe les mêmes réponses en fonction de la vitesse pour le modèle simulé et pour le robot réel.

## 6.5 Les capteurs au sol

Le modèle du capteur au sol s'est basé sur un type de papier dont la réponse des capteurs était le plus proportionnel au niveaux de gris théorique que sur les autres impressions (voir section 4).

**Rappel :** Ainsi le modèle suit une fonction du 1er degré définie par l'équation suivante :

$$g_v = -806.93 \cdot g_t + 853.24 \quad (18)$$

où  $g_t$  est le niveau de gris entre 0 et 1 (0 pour le blanc et 1 pour le noir). Le bruit est très faible et est défini par une constante valant  $\sigma_{bruit} = 0.45ua$ .

Le capteur au sol capte la lumière réfléchie sur une certaine zone (d'environ 1cm de rayon) sur la surface dont il effectue la moyenne.

L'implémentation s'est donc faite en appliquant un floutage de la texture au sol par l'application d'un filtre gaussien de taille 9x9.

Pour cela, on a créé une classe GroundSensor (voir Annexe F) qui est définie selon le code suivant :

```
// GroundSensor(Robot *owner, Vector pos, double noiseSd = 0.);
```

avec *\*owner* le pointeur sur le robot, *pos* la position du capteur sur le robot et *noiseSd* la constante définissant le bruit sur le signal du capteur.

La fonction *getValue* est codée de cette manière :

```
int GroundSensor::getValue(World* w)
{
    // compute absolute position and orientation
    const Matrix22 rot(owner->angle);
    absPos = owner->pos + rot * pos;
    double v = 0;

    // compute sensor value on a gaussian filtered ground
    for(int i = -4; i <= 4; i++)
    {
        for(int j = -4; j <= 4; j++)
        {
            double x = (double) i * 1 / 4;
            double y = (double) j * 1 / 4;
            Point pointtmp(absPos.x+x,absPos.y+y);
            Color color = w->getGroundColor(pointtmp);
            v += (- 806.93 * (3-(color.r() + color.g() + color.b())) / 3 + 853.24) *
                filter[i+4][j+4];
        }
    }
    // Adding constant noise before returning value
    return (int) gaussianRand(v, 0.45);
}
```

Le filtre est défini lors de l'initialisation du capteur au sol :

```
// Compute a gaussian filter
double sigma = 0.84089642;

int x, y;
for(y = -4; y <= 4; y++)
    for(x = -4; x <= 4; x++)
        filter[x + 4][y + 4] = 1. / (2 * M_PI * sigma * sigma) *
            exp(-(x * x + y * y) / (2 * sigma * sigma));
```

Similairement aux capteurs horizontaux, les capteurs au sol ont été initialisés dans le constructeur de la classe Thymio2 (dans Enki) :

```
|| groundsensor0(this, Vector(7.2, 1.2), 0.45),
|| groundsensor1(this, Vector(7.2, -1.2), 0.45)
```

La texture au sol se charge en modifiant les paramètres du fichier XML se chargeant à l'ouverture de Asebaplayground. Il suffit de définir la taille en largeur et hauteur de la texture ainsi que de lui donner le chemin de l'image à charger.

Exemple :

```
<world w="110.4" h="110.4" color="wall" wtexture="1024" htexture="1024"
      groundTex ="playgroundtexture.png" />
```

L'ajout du code effectuant la lecture du fichier XML afin de permettre de charger la texture au sol est défini comme suit :

```
// Create the world
QImage groundtext;

QDomElement worldE = domDocument.documentElement().firstChildElement("world");
Enki::Color worldColor(Enki::Color::gray);
if (!colorsMap.contains(worldE.attribute("color")))
    std::cerr << "Warning, world walls color " << worldE.attribute("color").
        toStdString() << " undefined\n";
else
    worldColor = colorsMap[worldE.attribute("color")];

// Getting texture and loading it to a QImage
if(worldE.hasAttribute("groundTex"))
{

    std::string str(worldE.attribute("groundTex").toStdString());
    QString qstr = QString::fromStdString(str);
    QFileinfo fileInfo(fileName);
    QDir groundtextdir = fileInfo.absoluteDir();
    if(groundtextdir.exists(qstr))
        groundtext = QImage(groundtextdir.filePath(qstr));
}

Enki::World world(
    worldE.attribute("w").toDouble(),
    worldE.attribute("h").toDouble(),
    worldColor,
    // Loading texture if it exists
    groundtext.width() ? worldE.attribute("wtexture").toDouble() : 0,
    groundtext.height() ? worldE.attribute("htexture").toDouble() : 0,
    groundtext.width() ? (uint32_t*)groundtext.constBits() : 0
);
```

Pour comparer le modèle simulé et le modèle réel, on a effectué des mesures sur le capteur lorsque les robots passent sur différents niveaux de gris : les impressions 1 qui ont été pris comme modèle de mesure pour réaliser le modèle simulé avec un robot réel (TH095), et une texture au sol avec les niveaux de gris entre 0 et 1. Les niveaux de gris sont espacé de 2.5cm sur la feuille pour le robot réel et dans le monde simulé pour le robot simulé (voir fig. 46 et 47).



Figure 46: Condition de mesure pour le robot TH095



Figure 47: Condition de mesure dans l'univers simulé

Et on applique le code aseba suivant :

```
var mesures_capteurs[6] = 0, 0, 0, 0, 0, 0, 0
timer.period[0] = 1
motor.left.target = 50
motor.right.target = 50

call leds.prox.v(0,0)

onevent timer0
mesures_capteurs = [prox.ground.ambiant[0], prox.ground.ambiant[1], prox.
    ground.reflected[0], prox.ground.reflected[1], prox.ground.delta[0], prox.
    ground.delta[1]]
emit capteurs mesures_capteurs
```

On obtient respectivement les figures 48 et 49 pour le robot TH095 et le modèle simulé.

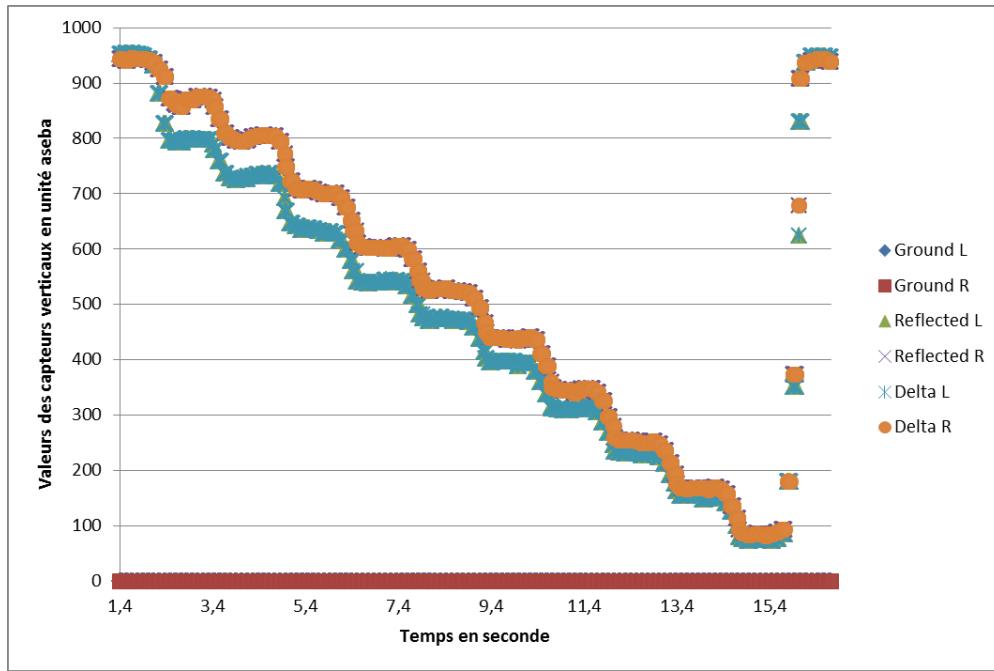


Figure 48: Résultat de mesure des capteurs au sol du Thymio TH095

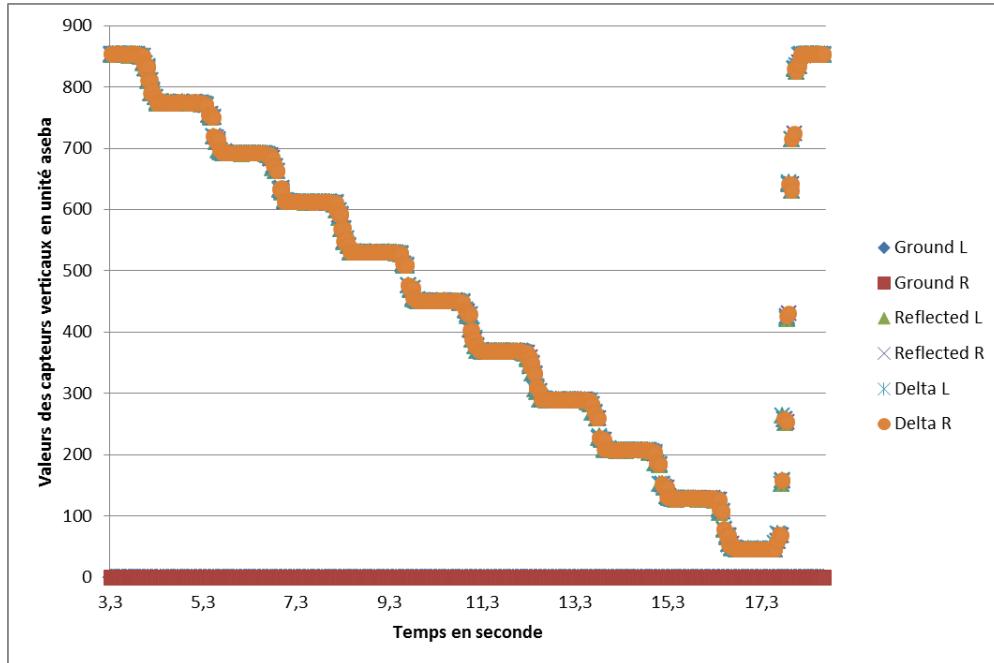


Figure 49: Résultat de mesure des capteurs au sol du modèle simulé

Les mesures sur le modèle simulé et le robot TH095 sont très semblables. Comme il n'y a qu'un seul modèle de capteur sur le modèle simulé, il n'y pas de grosse différence entre les mesures des 2 capteurs au sol mis à part le bruit qui est très faible. A l'inverse, les capteurs au sol du robot TH095 ne mesurent pas la même chose : ils ont une pente légèrement différente.

## 6.6 Les LEDs

Rappel : Des textures pour chaque LEDs sont superposées sur la texture de base afin de simuler l'activation des LEDs. Leur intensité est définie selon la transparence de l'image pour un alpha valant entre 80 et 255 correspondant à la plage d'intensité de 1 à 32 dans le code aseba lors de l'appel d'une fonction d'activation de LEDs.

Dans le code C++, on a créé une classe Thymio2Model (voir Annexe F) qui contient des fonctions de dessins du modèle 3D du Thymio II et de la mise à jour de la texture avec la superposition des textures de LEDs. On définit aussi les variables et la structure suivantes :

```
// Contains the texture of the body
QImage bodytexture;
// Contains the texture of the wheel
QImage wheeltexture;
// Contains the texture of the leds
QImage ledstexture[35];
    // To gain computing time, texture zone a defined between two points.
    // Operation on pixels would only be done between the defined zone
struct Texturezone {
    int x1;
    int y1;
    int x2;
    int y2;
};

// contains the texture zone for each leds
Texturezone tz[35];
```

Le tableau *tz[35]* contient la zone où se situe la LEDs sur l'ensemble de la texture pour permettre un gain de temps lors de la superposition de la texture de LED sur la texture du Thymio II. Les textures sont toutes initialisées dans la fonction *textureinit* :

```
void Thymio2Model::textureinit(void)
{
    bodytexture = QImage(":/textures/thymiobody.png");
    wheeltexture = QImage(":/textures/thymiowheel.png");
    ledstexture[0] = QImage(":/textures/leds_circle_0.png");
    ledstexture[1] = QImage(":/textures/leds_circle_1.png");
    ledstexture[2] = QImage(":/textures/leds_circle_2.png");
    ledstexture[3] = QImage(":/textures/leds_circle_3.png");
    ledstexture[4] = QImage(":/textures/leds_circle_4.png");
    ledstexture[5] = QImage(":/textures/leds_circle_5.png");
    ledstexture[6] = QImage(":/textures/leds_circle_6.png");
    ledstexture[7] = QImage(":/textures/leds_circle_7.png");
    ledstexture[8] = QImage(":/textures/leds_button_0.png");
    ledstexture[9] = QImage(":/textures/leds_button_1.png");
    ledstexture[10] = QImage(":/textures/leds_button_2.png");
    ledstexture[11] = QImage(":/textures/leds_button_3.png");
    ledstexture[12] = QImage(":/textures/leds_h_0.png");
    ledstexture[13] = QImage(":/textures/leds_h_1.png");
    ledstexture[14] = QImage(":/textures/leds_h_2.png");
    ledstexture[15] = QImage(":/textures/leds_h_3.png");
    ledstexture[16] = QImage(":/textures/leds_h_4.png");
    ledstexture[17] = QImage(":/textures/leds_h_5.png");
    ledstexture[18] = QImage(":/textures/leds_h_6.png");
    ledstexture[19] = QImage(":/textures/leds_h_7.png");
    ledstexture[20] = QImage(":/textures/leds_v_0.png");
    ledstexture[21] = QImage(":/textures/leds_v_1.png");
```

```

    ledstexture[22] = QImage(":/textures/leds_rc.png");
    ledstexture[23] = QImage(":/textures/leds_sound.png");
    ledstexture[24] = QImage(":/textures/leds_temperature.png");
    ledstexture[26] = QImage(":/textures/leds_top.png");
    ledstexture[29] = QImage(":/textures/leds_bottom_l.png");
    ledstexture[32] = QImage(":/textures/leds_bottom_r.png");
}

```

Lorsque qu'on appelle une fonction *call.leds* dans le code aseba, le flag *updateded* est mis à 1 et on effectue la superposition des LEDs concernées (le tableau *leds\_flags* indique quelles sont les LEDs qui ont été activées).

Dans le code suivant, on modifie seulement la transparence définie par la couche alpha de l'image pour les LEDs n'ayant qu'un seul paramètre propre (entre 0 et 32) : cela concerne les LEDs du cercle, des boutons, des capteurs, du récepteur à télécommande infrarouge et de l'indicateur de son. Cependant certaines zones du Thymio comme le dessus du Thymio sont constituées de LEDs multicolores permettant de configurer des couleurs spécifiques en fonction du niveau d'intensité des 3 paramètres de couleurs rouge(R), vert(G) et bleu(B).

On peut ainsi superposer les textures selon leur couche alpha (entre 80 et 255) définie par le niveau d'intensité (entrée dans l'appel de fonction de *leds* dans aseba et valant entre 0 et 32) ainsi que la couleur résultante pour les LEDs multicolores :

```

if(dw->updateded == true)
{
    // Reinitialize body texture
    textureinit();

    // LEDS : circle, buttons, prox, rc and sound
    for(int i = 0; i < 24; i++)
    {
        if(dw->leds_flag[i] == true)
        {
            for (int x = tz[i].x1; x <= tz[i].x2; x++)
            {
                for (int y = tz[i].y1; y <= tz[i].y2; y++)
                {
                    int alpha = qAlpha(ledstexture[i].pixel(x,y));
                    QColor color(ledstexture[i].pixel(x,y));

                    if (alpha != 0) {
                        float value = ((float)dw->leds[i] * (22./33.) + 10.) * alpha / 32;
                        color.setAlpha(value);
                    }
                    else
                        color.setAlpha(0);

                    ledstexture[i].setPixel(x, y, color.rgb());
                }
            }
            bodytexture = createImageWithOverlay(bodytexture, ledstexture[i]);
        }
    }

    ...
    // Top LEDs color parameters
    if(dw->leds_flag[26] == true)
    {
        for (int x = tz[26].x1; x <= tz[26].x2; x++)
    }
}

```

```

{
    for (int y = tz[26].y1; y <= tz[26].y2; y++)
    {
        int alpha = qAlpha(ledstexture[26].pixel(x,y));
        QColor color(ledstexture[26].pixel(x,y));
        int intensity;
        if(dw->leds[26] >= dw->leds[27] && dw->leds[26] >= dw->leds[28])
            intensity = dw->leds[26];
        else if (dw->leds[27] >= dw->leds[26] && dw->leds[27] >= dw->leds[28])
            intensity = dw->leds[27];
        else
            intensity = dw->leds[28];

        if (alpha != 0) {
            float value = ((float)intensity * (22./33.) + 10.) * alpha / 32;
            color.setAlpha(value);
            color.setRed((float)((float)dw->leds[26]/((float)dw->leds[26] + (float)
                dw->leds[27] + (float)dw->leds[28]))*255);
            color.setGreen((float)((float)dw->leds[27]/((float)dw->leds[26] +
                float)dw->leds[27] + (float)dw->leds[28]))*255);
            color.setBlue((float)((float)dw->leds[28]/((float)dw->leds[26] + (
                float)dw->leds[27] + (float)dw->leds[28]))*255);
        }
        else
            color.setAlpha(0);

        ledstexture[26].setPixel(x, y, color.rgba());
    }
}
bodytexture = createImageWithOverlay(bodytexture, ledstexture[26]);
}

...
}
}

```

**Remarque :** la démarche appliquée pour l’application de la LED multicolore sur le dessus du robot est aussi appliquée pour la LED de température et les LEDs des cotés gauche et droit.

Sur les figures 50, 51 et 52, on peut comparer le modèle simulé avec le robot TH095 lors de l’affichage des LEDs avec le code aseba suivant :

```

call leds.buttons(32,32,32,32)
call leds.prox.h(32,32,32,32,32,32,32,32)
call leds.prox.v(32,32)
call leds.top(32,0,0)
call leds.circle(32,32,32,32,32,32,32,32)
call leds.bottom.left(32,0,32)
call leds.bottom.right(0,32,32)
call leds.rc(32)
call leds.sound(32)
call leds.temperature(32,32)

```

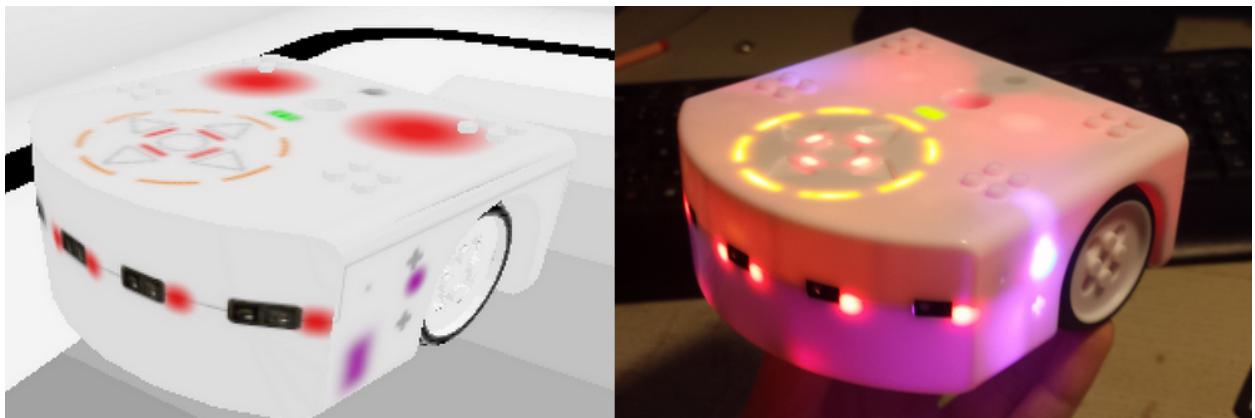


Figure 50: Comparaison de l'affichage des LEDs, vue sur l'avant et le dessus des robots



Figure 51: Comparaison de l'affichage des LEDs, vue sur l'avant et le dessus des robots

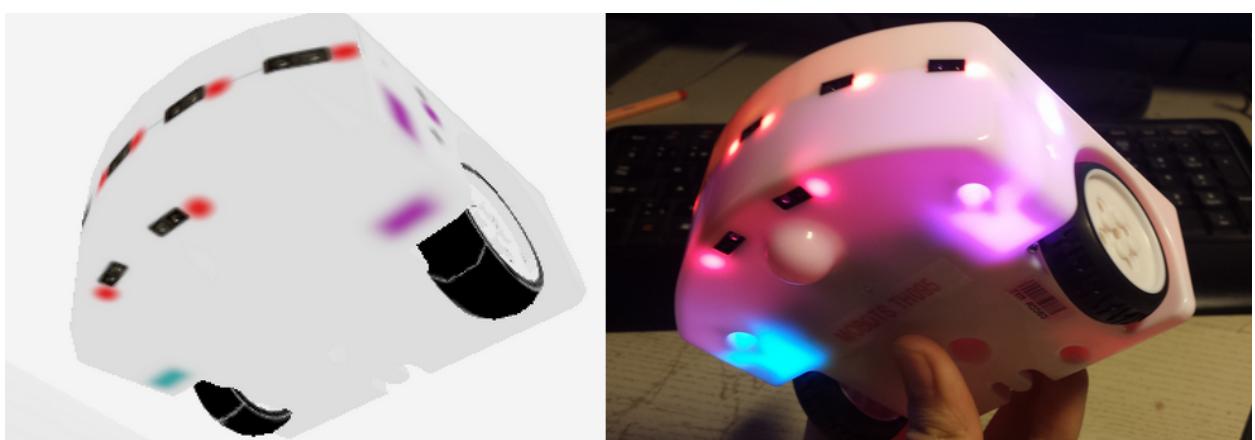


Figure 52: Comparaison de l'affichage des LEDs, vue sur l'avant et le dessus des robots

Les textures s'affichent correctement avec les bonnes couleurs (résultantes des mélanges de rouge vert et bleu).

Cette fois, on effectue le code suivant qui affiche les 6 LEDs des capteurs (sur l'avant du robot) avec des intensités différentes :

```
|| call leds.prox.h(0,5,10,15,25,32,0,0)
```

on obtient le résultat suivant sur la figure 53

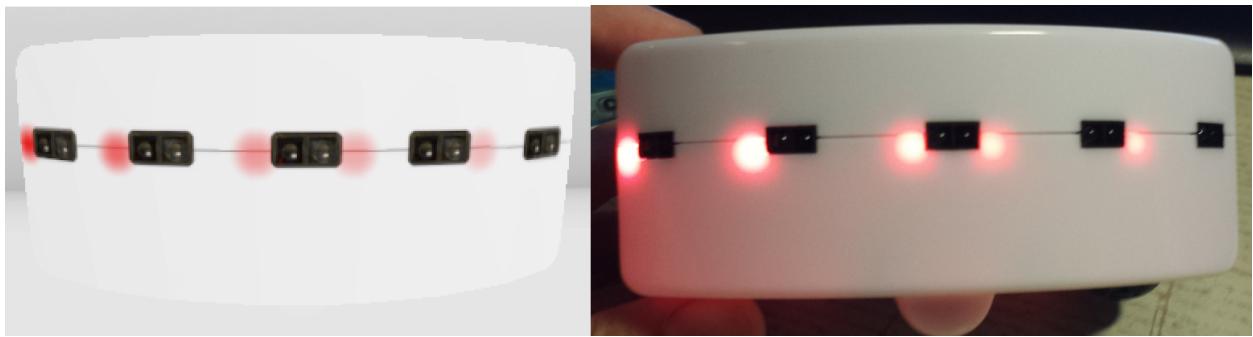


Figure 53: Comparaison de l'affichage des LEDs avant des capteurs sur le robot simulé et le Thymio TH095

Le rendu des textures n'est pas vraiment aussi beau que sur un robot réel mais la simulation de l'intensité et du rendu des couleurs est quant à elle fidèle à l'affichage des LEDs d'un robot réel.

## 7 Conclusion

La solution présentée dans ce rapport se base uniquement sur des mesures ayant été effectuées sur 3 robots Thymio II différents, et elle s'appuie sur des notions de physiques basiques (par exemple pour les moteurs) et principalement sur des notions avancées en codage C++.

Les mesures ont été séparées en 4 modules. Les capteurs horizontaux renvoient un signal dépendant de la distance avec un obstacle. Les mesures ont permis d'établir un modèle non trivial de la caractéristique du signal mesuré. Les mesures effectuées sur les moteurs ont permis d'établir d'autres modèles simples, et on a pu aussi constater que le bruit sur le signal de la vitesse et du PWM est très important et non négligeable. Les capteurs verticaux ont été utilisés pour une application différente de celle des capteurs horizontaux : la mesure du niveau de gris d'une surface (car la distance entre le capteur et le sol reste constante). Et enfin la simulation de l'affichage des LEDs s'est uniquement basée sur des observations et sur l'application de texture sur le modèle 3D.

Toute l'implémentation m'a permis de me familiariser avec les différentes bibliothèques de Qt ( QTimer, QImage, QDir, QString, etc.) et de surtout me familiariser d'avantage avec le codage en C++. J'ai pu aussi apprendre quelques bases en OpenGL tout en passant par de la modélisation d'objet 3D sous Blender (ce qui fut une expérience très enrichissante). Enfin, ce projet m'a permis d'adapter et d'établir des modèles en se basant sur des mesures rigoureuses des différents composants du robot Thymio II. Toutes ces connaissances m'ont permis de modéliser une première version du robot.

Cependant, certaines améliorations auraient pu être appliquées comme des paramètres de dispersions permettant de ne pas avoir deux capteurs, ou deux moteurs complètement identiques et ainsi de pas utiliser plusieurs robots ayant exactement les mêmes caractéristiques et les mêmes réponses de signaux.

La modélisation 3D est aussi loin d'être parfaite, en particulier le modèle visuel qui n'est pas aussi beau à contempler qu'un Thymio II réel dont toutes les LEDs sont allumées. Toutefois la modélisation de l'affichage des LEDs remplit le cahier des charges de manière rigoureuse.

D'un point de vue plus large, l'objectif réel de ce projet est de rendre possible l'utilisation d'un robot Thymio II simulé sans avoir recours à un robot physique. Ceci va permettre à des centaines (voire des milliers) d'utilisateurs d'avoir libre accès à un Thymio II simulé afin d'apprendre quelques bases de robotiques et de programmation.

Je tiens à remercier particulièrement :

- Philippe Retornaz pour son soutien et ses explications sur le Thymio II durant le semestre.
- Stéphane Magnenat pour m'avoir guidé tout au long du semestre et particulièrement concernant certains codages délicats en OpenGL.
- Fanny Riedo pour m'avoir accompagné, soutenu et aidé jusqu'au bout de la réalisation de ce projet.

Nicolas Dinh

## 8 Annexes

### 8.1 Annexe A : Code Matlab pour déterminer les paramètres optimaux pour les capteurs IR

```

1 valeurs = load('matlab_valeurs_capteurs.txt');
2 erreurmin(2:20) = 10E40;
3 x0_optimum(2:20) = 0;
4 m_optimum(2:20) = 0;
5 c_optimum(2:20) = 0;
6 for y=2:1:20
7     for c = 0:0.1:120
8         for x0=0:0.1:1
9             for m = 4000:1:5000
10                n = 0;
11                erreur = 0;
12
13                for x=1:1:12
14                    f = (m*(c-x0*x0))/(x*x - 2*x0*x+c);
15                    erreur = erreur + (f - valeurs(x,y))*(f - valeurs(x,y));
16                    n = n + 1;
17                end
18
19                erreur = erreur / n;
20
21                if (erreur < erreurmin(y))
22                    erreurmin(y) = erreur;
23                    c_optimum(y) = c;
24                    x0_optimum(y) = x0;
25                    m_optimum(y) = m;
26                end
27
28            end
29        end
30    end
31 end
32 c_optimum
33 erreurmin
34 x0_optimum
35 m_optimum

```

## 8.2 Annexe B : Résultats du code Matlab pour les paramètres des capteurs IR $x_0$ , $m$ et $c$

Capteurs	c	m	x0
TH085-1	47.4	4611	0.033
TH085-2	63.4	4411	0.009
TH085-3	84.6	4634	0.024
TH085-4	58.1	4090	0.012
TH085-5	61.9	4625	0.015
TH085-6	60.7	4597	0.021
TH091-0	53.4	4232	0.033
TH091-1	100.4	4552	0.033
TH091-2	109.5	4571	0.009
TH091-3	95.5	4646	0.024
TH091-4	104.1	4613	0.012
TH091-5	101.5	4423	0.015
TH091-6	74.2	4595	0.021
TH095-0	67.7	4630	0.021
TH095-1	74	4548	0.033
TH095-2	85.1	4672	0.009
TH095-3	46.2	4136	0.024
TH095-4	85.2	4613	0.012
TH095-5	41.6	4399	0.015
<b>Moyenne</b>	<b>73.3</b>	<b>4505</b>	<b>0.03</b>

Le capteur 0 du robot TH085 et le capteur 6 du robot TH095 ont été écartés pour cause de défaillance.

## 8.3 Annexe C : Tableau de calcul du retard $\tau$ (en seconde) pour le profil de vitesse des moteurs

Vitesse	TH085	TH091	TH095
50	0.015	0.021	0.033
100	0.012	0.012	0.009
150	0.015	0.012	0.024
200	0.015	0.03	0.012
250	0.015	0.021	0.015
300	0.015	0.018	0.012
350	0.024	0.024	0.018
400	0.021	0.012	0.021
450	0.021	0.024	0.018
500	0.033	0.027	0.021

## 8.4 Annexe D : Tableau de calcul du retard $\tau$ (en seconde) pour le profil de courant des moteurs

Robot :		TH085		TH091		TH095	
Consigne		Tau G	Tau D	Tau G	Tau D	Tau G	Tau D
50		0.527	0.519	0.601	0.531	0.504	0.437
100		0.547	0.566	0.534	0.53	0.594	0.684
150		0.618	0.621	0.549	0.485	0.525	0.522
200		0.627	0.618	0.552	0.535	0.633	0.692
250		0.464	0.476	0.486	0.498	0.592	0.53

## 8.5 Annexe E : Liste des fichiers modifiés ou ajoutés dans les bibliothèques Enki et Aseba

Dans la bibliothèque Enki :

- **enki/CMakeLists.txt** : ajout des nouveaux fichiers .cpp
- **enki/PhysicalEngine.cpp** et **PhysicalEngine.h** : ajout de la fonction *setThymio2Hull*
- **enki/interactions/GroundSensor.cpp** et **GroundSensor.h** : nouveaux fichiers calculant le niveau de gris de la texture au sol
- **enki/interactions/IRSensor.cpp** : légère modification de la fonction *finalize* permettant de renvoyer 0 sans aucun bruit si l'obstacle est hors de portée.
- **enki/robots/DifferentialWheeledThymio2.cpp** et **DifferentialWheeledThymio2.h** : nouveaux fichiers calculant les variables mesurées sur le moteur des robots Thymio II
- **enki/robots/thymio2/Thymio2.cpp** et **Thymio2.cpp.h** : nouveaux fichiers définissant la classe d'un Thymio II simulé et initialisant les paramètres dont il dépend (moteurs et capteurs)
- **viewer/CMakeLists.txt** : ajout des nouveaux fichiers .cpp
- **viewer/EpuckModel.cpp** et **EpuckModel.h** : ajout de la fonction virtuelle *updatetexture*
- **viewer/MarxbotModel.cpp** et **MarxbotModel.h** : ajout de la fonction virtuelle *updatetexture*
- **viewer/Thymio2Model.cpp** et **Thymio2Model.h** : nouveaux fichiers dessinant le robot Thymio II
- **viewer/Viewer.cpp** et **Viewer.h** : ajout de la fonction virtuelle *updatetexture*. Ajout du Thymio2Model.h ainsi que l'initialisation et ajout de enki/robots/thymio2/thymio2.h dans les headers. Ajout de l'appel de updatetexture dans la fonction *paintGL()*.
- **viewer/objects/Thymio2Body.cpp** et **Thymio2Wheel.cpp** : nouveaux fichiers contenant les données OpenGL (faces, sommets, textures) du Thymio II
- **viewer/objects/Objects.h** : ajout de Thymio2Body et Thymio2Wheel dans la liste des variables OpenGL

- **viewer/textures** : tous les fichiers png de textures de LEDs et de textures du Thymio
- **viewer/enki-viewer-textures.qrc** : ajout des fichiers textures dans les qressources

Dans le dossier Aseba :

- **targets/challenge/challenge.cpp** : ajout de la fonction virtuelle updatetexture
- **targets/playground/CMakeLists.txt** : ajout des nouveaux fichiers .cpp
- **targets/playground/unifr2.playground** : Terrain de jeu prédéfini pour le Thymio II
- **targets/playground/Thymio2-description.c** : nouveau fichier contient les descriptions des variables, des fonctions natives et des évènements
- **targets/playground/Thymio2.cpp** et **Thymio2.h** : nouveaux fichiers contenant les variables, les fonctions natives et les évènements du robot simulés pour Aseba.
- **targets/playground/playgroundtexture.png** : texture utilisé pour le terrain de jeu prédéfini.
- **targets/playground/playground.cpp** : ajout de la lecture d'un fichier image pour l'attribuer à la classe World en tant que texture au sol. Ajout de la lecture d'éventuels robots Thymio II dans le fichier xml.
- **targets/playground/playground.cpp** : ajout du header "thymio2.h" et ajout de la ligne :

```
|| managedObjectsAliases[&typeid(AsebaThymio2)] = &typeid(Thymio2);
```

## 8.6 Annexe F : Définitions des nouvelles classes

Dans le fichier aseba/targets/playground/thymio2.h :

La classe ThymioTimer définit un QTimer qui sera utilisé pour l'activation des évènements du Thymio à travers la fonction timer\_update() qui est appelée à la fréquence définie lors de l'initialisation du timer :

```
class ThymioTimer : public QObject
{
    Q_OBJECT

public:
    QTimer timer;
    ThymioTimer(QObject *parent = 0, AsebaThymio2 *thymio = 0);
    virtual ~ThymioTimer();

public slots:
    void timer_update();

private:
    AsebaThymio2 *thymioparent;
};
```

La classe AsebaThymio2 définit toutes les variables, fonctions natives et événements du Thymio2 et elle hérite de la classe Thymio2 définie dans Enki :

```

class AsebaThymio2 : public Thymio2, public Aseba::AbstractNodeGlue, public
    Aseba::SimpleDashelConnection
{

public:
    AsebaVMState vm;
    std::valarray<unsigned short> bytecode;
    std::valarray<signed short> stack;
    World* w;

struct Variables
{
    sint16 id;
    sint16 source;
    sint16 args[32];
    sint16 productId;
    sint16 fwversion[2];
    sint16 robottype;

    sint16 buttonraw[5];
    sint16 buttonbackward;
    sint16 buttonleft;
    sint16 buttoncenter;
    sint16 buttonforward;
    sint16 buttonright;
    sint16 buttonmean[5];
    sint16 buttonnoise[5];

    sint16 proxhorizontal[7];
    sint16 proxgroundambient[2];
    sint16 proxgroundreflected[2];
    sint16 proxgrounddelta[2];

    sint16 motorlefttarget;
    sint16 motorrighttarget;
    sint16 vbat[2];
    sint16 imot[2];
    sint16 motorleftspeed;
    sint16 motorrightspeed;
    sint16 motorleftpwm;
    sint16 motorrightpwm;

    sint16 acc[3];

    sint16 temperature;

    sint16 rc5adress;
    sint16 rc5command;

    sint16 micintensity;
    sint16 micthreshold;
    sint16 micmean;

    sint16 timerperiod[2];

    sint16 freeSpace[512];
}

```

```
    } variables;

public:
    AsebaThymio2(unsigned port, int id);
    virtual ~AsebaThymio2();
    void timer_update();

    // from THymio2

    virtual void controlStep(double dt);

    // from AbstractNodeGlue

    virtual const AsebaVMDescription* getDescription() const;
    virtual const AsebaLocalEventDescription * getLocalEventsDescriptions()
        const;
    virtual const AsebaNativeFunctionDescription * const *
        getNativeFunctionsDescriptions() const;
    virtual void callNativeFunction(uint16 id);

private:
    unsigned int event_flags;

    enum Event
    {
        EVENT_B_BACKWARD = 0,
        EVENT_B_LEFT,
        EVENT_B_CENTER,
        EVENT_B_FORWARD,
        EVENT_B_RIGHT,
        EVENT_BUTTONS,
        EVENT_PROX,
        EVENT_TAP,
        EVENT_ACC,
        EVENT_MIC,
        EVENT_SOUND_FINISHED,
        EVENT_TEMPERATURE,
        EVENT_RC5,
        EVENT_MOTOR,
        // Must be consecutive
        EVENT_TIMER0,
        EVENT_TIMER1,

        // Maximum count: 32
        EVENT_COUNT // Do not touch
    };

    ThymioTimer *timer;

    //int period[2];
    int counter;
    //TODO ajouter timer pour les evenements
};
```

**Dans le fichier enki/interactions/GroundSensor.h :**

La classe GroundSensor définit les variables et les fonctions nécessaires pour le calcul de la réponse du capteur par rapport au niveau de gris de la texture au sol :

```
class GroundSensor : public LocalInteraction
{
protected:
    ///! Absolute position in the world, updated on init()
    Vector absPos;
    ///! Relative position on the robot
    const Vector pos;
    ///! Standard deviation of Gaussian noise in the response space
    const double noiseSd;
    ///! Ground image which has been filtered by a gaussian filter
    double filter[9][9];

public:
    ///! Constructor
    /*!
     \param owner robot which embeds this sensor
     \param pos relative position (x,y) on the robot
     \param noiseSd standard deviation of Gaussian noise in the response space
     */
    GroundSensor(Robot *owner, Vector pos, double noiseSd = 0.);
    ///! Reset distance values
    int getValue(World* w);
    Point getAbsolutePosition(void) const { return absPos; }
};
```

**Dans le fichier enki/robots/DifferentialWheeledThymio2.h :**

La classe DifferentialWheeledThymio2 définit les variables et les fonctions nécessaires pour le calcul des vitesses, des PWM et des courants des moteurs du Thymio 2 afin de calculer la vitesse tangentielle et angulaire de l'objet physique pour la simulation :

```
class DifferentialWheeledThymio2: public Robot
{
public:
    ///! Left speed of the robot
    double leftSpeedTarget;
    double previousleftSpeedTarget;
    ///! Right speed of the robot
    double rightSpeedTarget;
    double previousrightSpeedTarget;

    ///! Real speed of the robot
    double leftSpeed;
    double rightSpeed;

    ///! PWM of the robot
    double leftPWM;
    double rightPWM;

    ///! Current of the robot
    double leftCurrenti;
    double rightCurrenti;
```

```


    /**
     * The encoder for left wheel; this is not a real encoder, but rather the
     * physical leftSpeed
    double leftEncoder;
    /**
     * The encoder for right wheel; this is not a real encoder, but rather the
     * physical rightSpeed
    double rightEncoder;
    /**
     * The odometry (accumulation of encoders) for left wheel
    double leftOdometry;
    /**
     * The odometry (accumulation of encoders) for right wheel
    double rightOdometry;

private :

    /**
     * Time counter
long int n, m;

    /**
     * Limits for speed
double leftnewlimitSpeed;
double leftpreviouslimitSpeed;
double rightnewlimitSpeed;
double rightpreviouslimitSpeed;

    /**
     * Limits for PWM
double leftnewlimitPWM;
double leftpreviouslimitPWM;
double rightnewlimitPWM;
double rightpreviouslimitPWM;

    /**
     * Limits for Current
double leftnewlimitCurrent;
double leftpreviouslimitCurrent;
double rightnewlimitCurrent;
double rightpreviouslimitCurrent;

protected:
    /**
     * Distance between the left and right driving wheels
double distBetweenWheels;
    /**
     * Maximum speed wheels can provide
double maxSpeed;
    /**
     * Relative amount of motor noise
double noiseFactor;

double noiseBase;

private:
    /**
     * Resulting angular speed from wheels
double cmdAngSpeed;
    /**
     * Resulting tangent speed from wheels
Vector cmdSpeed;

public:
    /**
     * Constructor
DifferentialWheeledThymio2(double distBetweenWheels, double maxSpeed, double
noiseFactor, double noiseBase);

    /**
     * Reset the encoder. Should be called when robot is moved manually.
     * Odometry is cleared too.
void resetEncoders();


```

```


//! Set the real speed of the robot given leftSpeed and rightSpeed. Add
noise. Update encoders.
virtual void controlStep(double dt);
//! Consider that robot wheels have immobile contact points with ground,
and override speeds. This kills three objects dynamics, but is good
enough for the type of simulation Enki covers (and the correct solution
is immensely more complex)
virtual void applyForces(double dt);
};


```

### Dans le fichier enki/robots/thymio2/thymio2.h :

La classe Thymio2 définit les variables des moteurs, des capteurs et des LEDs du robot :

```


class Thymio2 : public DifferentialWheeledThymio2
{
public:
    //! The infrared sensor 0 (front-left-left)
    IRSensor infraredSensor0;
    //! The infrared sensor 1 (front-left)
    IRSensor infraredSensor1;
    //! The infrared sensor 2 (front-front)
    IRSensor infraredSensor2;
    //! The infrared sensor 3 (front-right)
    IRSensor infraredSensor3;
    //! The infrared sensor 4 (front-right-right)
    IRSensor infraredSensor4;
    //! The infrared sensor 5 (back-left)
    IRSensor infraredSensor5;
    //! The infrared sensor 6 (back-right)
    IRSensor infraredSensor6;

    //! The ground sensor 0 (left)
    GroundSensor groundsensor0;
    //! The ground sensor 1 (right)
    GroundSensor groundsensor1;

    // Leds Parameters
    int leds[35]; // leds value (between 0 and 32)
    bool leds_flag[35]; // leds activation (which leds are activated)
    bool updatedled; // update flag (detect if there is a change for leds
texturing)

public:
    //! The bot's capabilities. You can simply select a predefined set of
    sensors. These correspond to the different extension modules that exist
    for the E-Puck.
    enum Capabilities
    {
        //! No sensor: not very useful
        CAPABILITY_NONE = 0,
        //! Basic_Sensors: Just the 7 IRSensors of the base module
        CAPABILITY_BASIC_SENSORS = 0x1
    };

public:
    //! Create a Thymio with certain modules aka capabilities (basic)
    Thymio2(unsigned capabilities = CAPABILITY_BASIC_SENSORS);


```

```

    //! Destructor
    ~Thymio2();
};


```

### Dans le fichier enki/viewer/Thymio2Model.h :

La classe Thymio2Model définit les variables nécessaires pour le dessin 3D du robot (ainsi que les textures des LEDs) dans le viewer :

```

class Thymio2Model : public ViewerWidget::CustomRobotModel
{
public:
    Thymio2Model(ViewerWidget* viewer);
    virtual void cleanup(ViewerWidget* viewer);
    virtual void draw(PhysicalObject* object) const;
    // Will update the texture (with different LEDs) if some LEDs are activated
    virtual void updatetexture(PhysicalObject* object, ViewerWidget* viewer);

private:
    // Contains the texture of the body
    QImage bodytexture;
    // Contains the texture of the wheel
    QImage wheeltexture;
    // Contains the texture of the leds
    QImage ledstexture[35];

    // To gain computing time, texture zone a defined between two points.
    // Operation on pixels would only be done between the defined zone
    struct Texturezone {
        int x1;
        int y1;
        int x2;
        int y2;
    };

    // contains the texture zone for each leds
    Texturezone tz[35];

private:
    // will re-initialize the QImage
    void textureinit(void);
    // will set the parameters for the texturezone in tz[i]
    void texturezoneinit(int i, int a, int b, int c, int d);
};


```

## 8.7 Annexe G : Mesures complémentaires pour les capteurs horizontaux

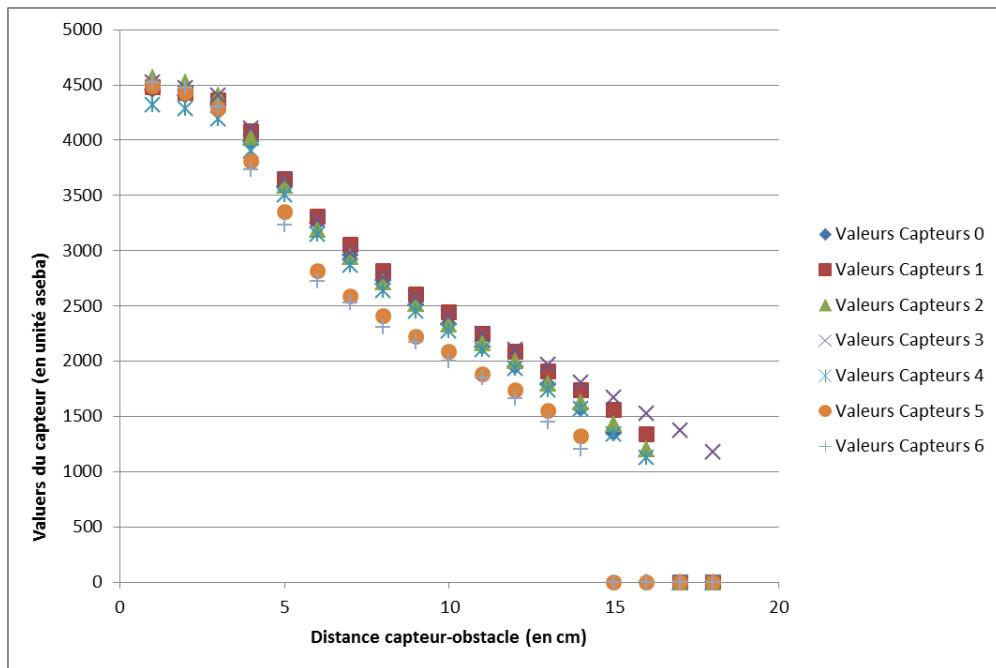


Figure 54: Mesure de la valeur des capteurs (en unité aseba) du robot Thymio II TH085 en fonction de la distance (en cm)

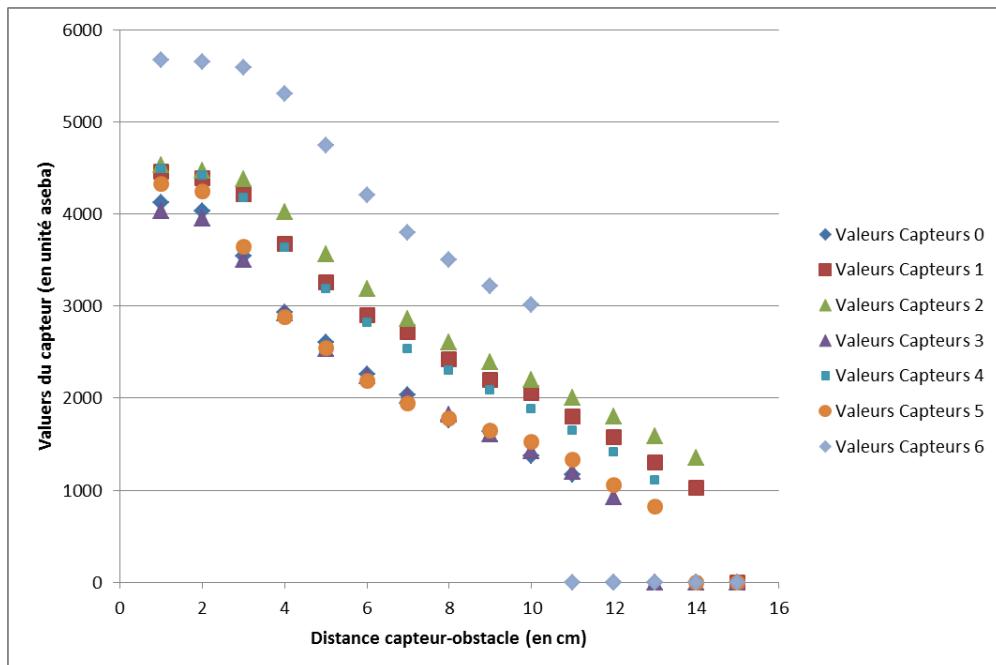


Figure 55: Mesure de la valeur des capteurs (en unité aseba) du robot Thymio II TH091 en fonction de la distance (en cm)

## 8.8 Annexe H : Mesures complémentaires pour les capteurs verticaux

Les mesures ont été faites sur des feuilles ayant été imprimées sur différents appareils d'impressions et faisant ressortir les niveaux de gris différemment. Les figures ci-dessous mettent en évidence cette différence.

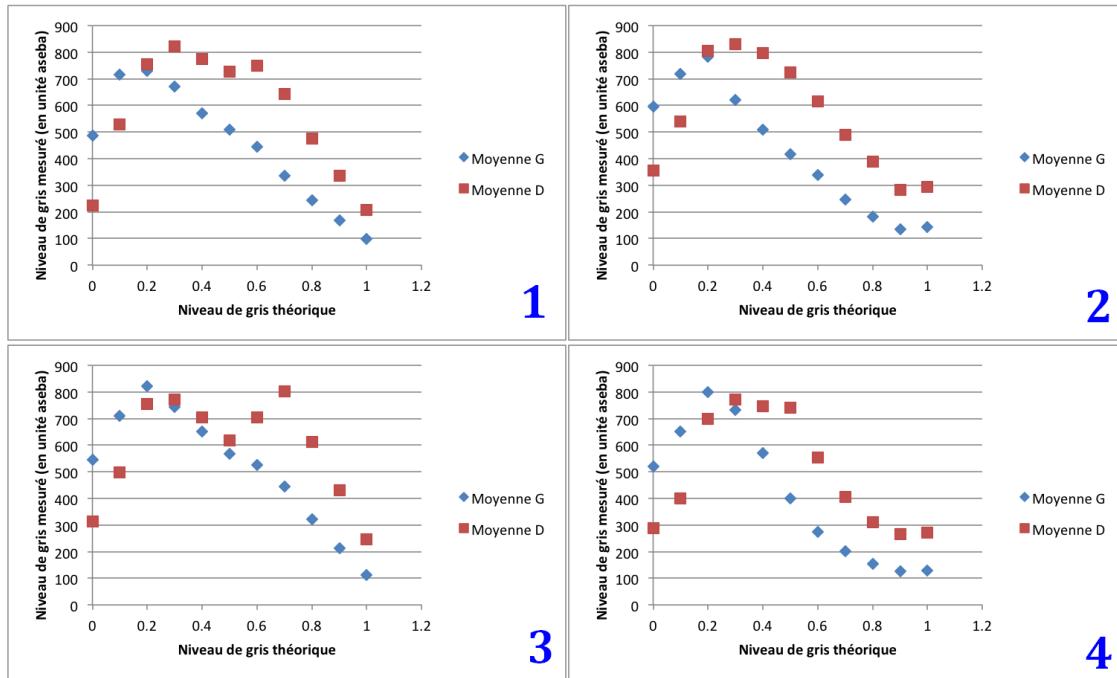


Figure 56: Différence entre les valeurs mesurées par les capteurs en unité aseba en fonction du niveau de gris théorique (entre 0 et 1) pour les 4 impressions avec le robot TH085

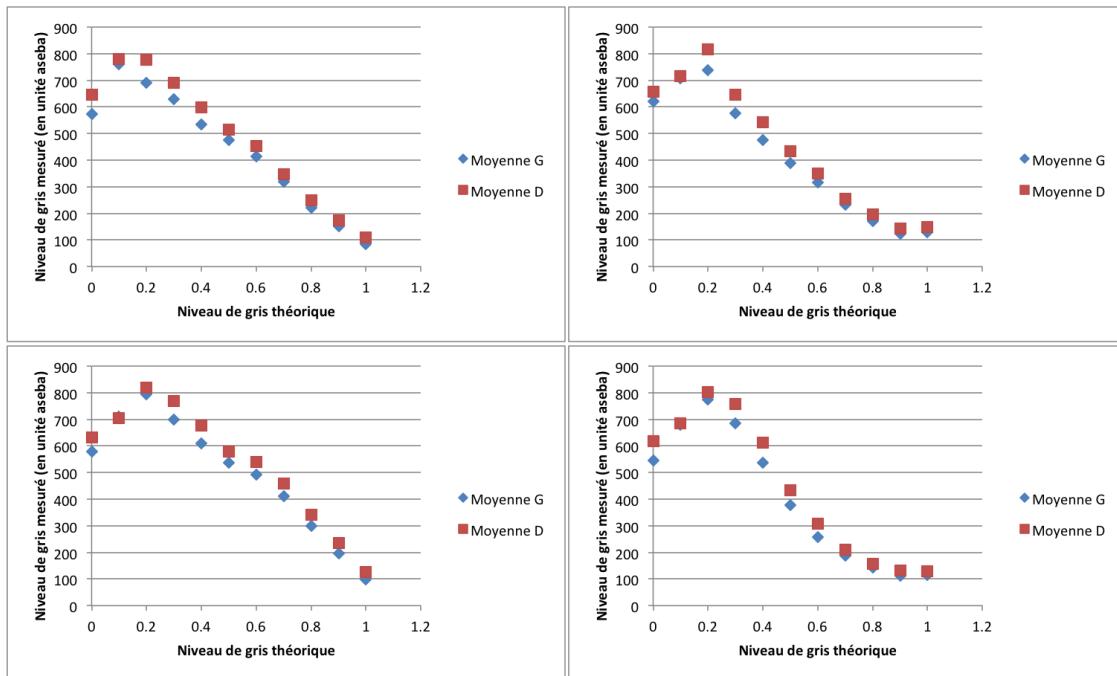


Figure 57: Différence entre les valeurs mesurées par les capteurs en unité aseba en fonction du niveau de gris théorique (entre 0 et 1) pour les 4 impressions avec le robot TH085