RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Ecole Nationale Polytechnique

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

Département d'électronique

Mémoire de projet de fin d'études

Pour l'obtention du diplôme d'Ingénieur d'état en électronique

Model Based Design pour l'implémentation D'un modèle Moto sur une cible embarquée (STM32). Application à un simulateur Moto.

Abdelghafour SID

Sous la direction de
Mr. R. SADOUN

Présenté et soutenu publiquement le 21/06/2017

Composition du Jury :

Président	M. TAGHI	MAA	Ecole Nationale Polytechnique
Promoteur	R. SADOUN	MCA	Ecole Nationale Polytechnique
Examinateur	M. ADNANE	PhD	Ecole Nationale Polytechnique

ENP 2017

RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Ecole Nationale Polytechnique

Département d'électronique

Mémoire de projet de fin d'études

Pour l'obtention du diplôme d'Ingénieur d'état en électronique

Model Based Design pour l'implémentation D'un modèle Moto sur une cible embarquée (STM32). Application à un simulateur Moto.

Abdelghafour SID

Sous la direction de
Mr. R. SADOUN

Présenté et soutenu publiquement le 21/06/2017

Composition du Jury :

Président	M. TAGHI	MAA	Ecole Nationale Polytechnique
Promoteur	R. SADOUN	MCA	Ecole Nationale Polytechnique
Examinateur	M. ADNANE	PhD	Ecole Nationale Polytechnique

ENP 2017

ملخص :

الهدف من هذا العمل هو تثبيت نموذج دراجة نارية في متحكم (STM32/LPC1768) وذلك باتباع طريقة التصميم استنادا لنموذج. باستعمال MATLAB / Simulink لتطوير النموذج و محاكاة و استعمال Embedded Coder لتوليد نصوص C/C++ الخاصة بهذا النموذج

الكلمات الدالة

Model-Based Design, modèle Moto, STM32, LPC1768, MATLAB, Simulink, Embedded Coder, Code C/C++.

Abstract

The aim of this work is the implementation of a model Motorcycle on an embedded target (STM32 / LPC1768), using the Model-Based Design process.

MATLAB / Simulink represents the model development and simulation tool and the "Embedded Coder" tool used to generate C / C ++ code.

Mots clefs : Model-Based Design, modèle Moto, STM32, LPC1768, MATLAB, Simulink, Embedded Coder, Code C/C++.,

Résumé

Le but de ce travail est l'implémentation d'un modèle Moto sur une cible embarqué (STM32/LPC1768), en utilisant le processus Model-Based Design.
MATLAB/Simulink représente l'outil de développement du modèle et de simulation et l'outil "Embedded Coder" utilisé pour la génération de Code C/C++.

Mots clés : Model-Based Design, modèle Moto, STM32, LPC1768, MATLAB, Simulink, Embedded Coder, Code C/C++.

Remerciement

En premier lieu, je tiens à remercier Dieu, de ma avoir aidé et de m'avoir donné la patience et lafoie pour finaliser ce mémoire.

Au terme de ce travail, je tiens à exprimer ma profonde gratitude et mes sincères remerciements à mon encadreur Mr Rabah SADOUN pour tout le temps qu'il m'a consacré, ses directives précieuses, et pour la qualité de son suivi tout au long du projet.

Mes plus vifs remerciements s'adressent aussi à tout le cadre Professeurs et administratifs de L'école National Polytechnique Alger

Mes remerciements vont en à toute personne qui a contribué de près ou de loin à l'élaboration de ce travail.

Abdelghafour SID

Table des matières

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ABREVIATIONS

MBD: Model-Based Design
2RM:Deux Roues Motorisées
CAN: Controller Area Network
TDD: Test Driven-Development
COTS: Commercial-of-the-Shelf
IID: Iterative Incremental Development
IFSTTAR : Institut français des sciences et technologies des transports, de l'aménagement et des réseaux
MIL: Model in the loop
PIL: Software-in-the-loop
SIL: Processor-in-the-loop
HIL: Hardware-In-the-Loop

Introduction Générale

Au cours de la dernière décennie, la densité de trafic urbain et les contraintes et les coûts croissants supportés par les conducteurs de voiture (stationnement, carburant, etc.) ont entraîné une augmentation de l'utilisation des véhicules à deux roues, même s'il est connu comme l'un des moins sécurisés.

L'étude d'un véhicule à deux roues a été négligée depuis longtemps alors que le nombre de motocyclistes morts a malheureusement augmenté. Pour cette raison, les établissements de recherche s'efforcent dès que possible d'apporter des réponses adaptées au problème de la sécurité des motocyclistes en développant des travaux sur la moto dont le risque d'accident fatal est beaucoup plus élevé par rapport aux véhicules quatre roues.
Les simulateurs de conduite sont des outils efficaces destinés à réaliser les différentes études dans un environnement sécurisé. Son objectif est de reproduire les sensations de conduite dans un environnement restreint et permet de former les novices.

Il permet de reconnaître les situations d'accident et d'évaluer les actions d'un motocycliste nécessaire pour reprendre le contrôle de son véhicule. Ainsi, il constitue un outil sûr pour la sensibilisation et la formation aux manœuvres d'urgence (freinage / évitement). En outre, il peut être utilisé pour l'étude du comportement de conduite des motards et pour la validation avec des observations effectuées dans des situations de conduite réelles.

Nous nous intéressons à la partie électronique embarquée d'un simulateur de conduite qui consiste en deux parties essentielles, qui sont intégrées dans un PC de table et les outils utilisés pour le développement et l'implémentation qui sont MATLAB/SIMULINK.

Dans les processus de développement des systèmes embarqué MATLAB est principalement utilisé pour le développement d'algorithmes, l'analyse de données et la visualisation avancée, tandis que son homologue SIMULINK est utilisé pour générer des modèles graphiques, effectuer des simulations de temps, faire des modélisations multi-domaine et effectuer des tests et des validations de systèmes.
Après la modélisation et la validation du système, on implémente ce système sur une cible hardware. La traduction manuelle ce modèle en langage C ou C++ est pénible et elle prend beaucoup de temps.

Récemment, une méthodologie de développement connue sous le nom de Model-Based Design (MBD) est apparue. Dans MBD, un modèle de système constitue le cœur du cycle de développement complet, de la définition des spécifications fonctionnelles aux phases de test et de validation par la génération du code de mise en œuvre. Par rapport aux processus de développement traditionnels, MBD réduit le temps de conception, les erreurs de codage, permet des tests de validation efficaces tout en réduisant les coûts de développement.

L'objectif général de ce travail est d'appliquer le processus MBD pour implémenter la première partie de simulateur de conduite (modèle dynamique) sur un STM32F303K8 et implémenter la deuxième partie(le modèle de la plateforme) sur un ARM LPC1768 tout en évaluant leurs performances.

Chapitre I : Méthodologies de conception

I-1 Introduction

Nous présentons dans ce présent chapitre le contexte de l'outil auquel nous aurons recours pour le développement d'une application embraquée sur des cibles technologiques bas coût. Pour rappel, cette application vise à exécuter les deux modèles clefs du simulateur d'IFSTAR.

Pour y arriver, nous passerons en revue quelques approches (ou méthodes) dédiées au développement d'applications embarquées.

I-2 Les systèmes embarqués

Les systèmes embarqués sont des systèmes de traitement d'information et de commande Intégrés dans un processus physique. De ce fait, ils doivent assurer la disponibilité des ressources aussi bien mémoire que de calcul et de communication tenant compte des contraintes temporelles imposées.. Les performances de ces systèmes ne cessent de progresser grâce au développement des systèmes électroniques qui se traduit par plus de puissance de calcul par unité de volume à moindre coût avec de plus en plus une faible consommation d'énergie. Les domaines d'utilisation de ces systèmes sont de plus en plus variés. On peut citer pour l'exemple l'aérospatial, l'aéronautique, l'électroménager, l'environnement, les équipements médicaux et le transport.

Les systèmes embarqués peuvent être développés en intégrant des composants ou des systèmes
Électroniques commerciaux grand "public". On parle alors de systèmes embarqués basés *COTS (Commercial-of-the-Shelf)* avec des avantages substantiels en termes de réduction du coût.
La large diffusion de ces circuits *COTS* assure un retour d'expériences enrichissant en termes de méthodes d'utilisation et d'intégration ainsi que leur validation grâce aux tests à grande échelle effectués par une communauté d'utilisateurs.

La réduction du temps de développement devient alors un autre critère crucial devant aboutir à des temps de mise sur le marché (Time to Market) qui doivent être favorable.
Le choix d'une méthode de développement devient aussi crucial.

I-2.1 Contraintes liées à l'usage des systèmes embarqués

Les systèmes embarqués sont soumis à des contraintes en termes d'exécution temps-réel, de fiabilité, de sécurité et de disponibilité tout en ayant des limitations sur le plan de la puissance électrique embarquée, de la taille, du poids et les capacités de calcul. La limitation de la puissance de calcul nécessite l'optimisation du code au détriment de sa

clarté, de sa portabilité et de sa modularité avec des interfaces homme-machine (IHM) limitées.

Les contraintes économiques, comme le coût, le temps de mise sur le marché (time-to-market), la flexibilité pour répondre aux exigences des clients, les interactions entre les parties prenantes, la durabilité et la gestion des COTS impliquent des aspects organisationnels importants d'où l'émergence de l'ingénierie des systèmes embarqués.

I.3 Ingénierie des systèmes embarqués

La forte dualité entre les systèmes embarqués et les systèmes informatiques classique induit une similarité dans les approches de l'organisation du développement et les méthodes qui y sont associées avec cette particularité que les systèmes embarqués peuvent être soumis à des contraintes plus strictes. Ce qui rend leur développement un processus complexe. Cette complexité de développement est traitée dans le cadre de l'ingénierie des systèmes en proposant des modèles des cycles de vie qui définissent un ensemble d'activités, les entrées et sorties de ces activités, les rôles et les responsabilités, les outils, ainsi que les délais et les coûts de développement.

La plupart des cycles de vie des systèmes embarqués sont une adaptation des processus de développement des systèmes logiciels classiques qui commencent par l'analyse des besoins et des spécifications jusqu'au déploiement en passant par la conception, l'implémentation, l'intégration et le test du système.
On peut citer trois modèles de cycle de développement :

1-3.1 Modèle en cascade

Constitue l'approche naturelle et historiquement la première mise en œuvre
D'une approche méthodologique de développement.
Elle consiste en la phase d'analyse avant la phase d'implantation.
Le modèle en cascade (Figure I.1) décrit une succession d'étapes. Même si on l'étang avec des possibilités de retour en arrière, idéalement limitées à la seule phase qui précède celle remise en cause, le développement reste fondamentalement linéaire. En particulier, il se fonde sur l'hypothèse souvent irréaliste que l'on peut dès le départ définir complètement et en détail ce qu'on veut réaliser. Une variante améliorée consiste en des itérations locales au sein de chaque étape.

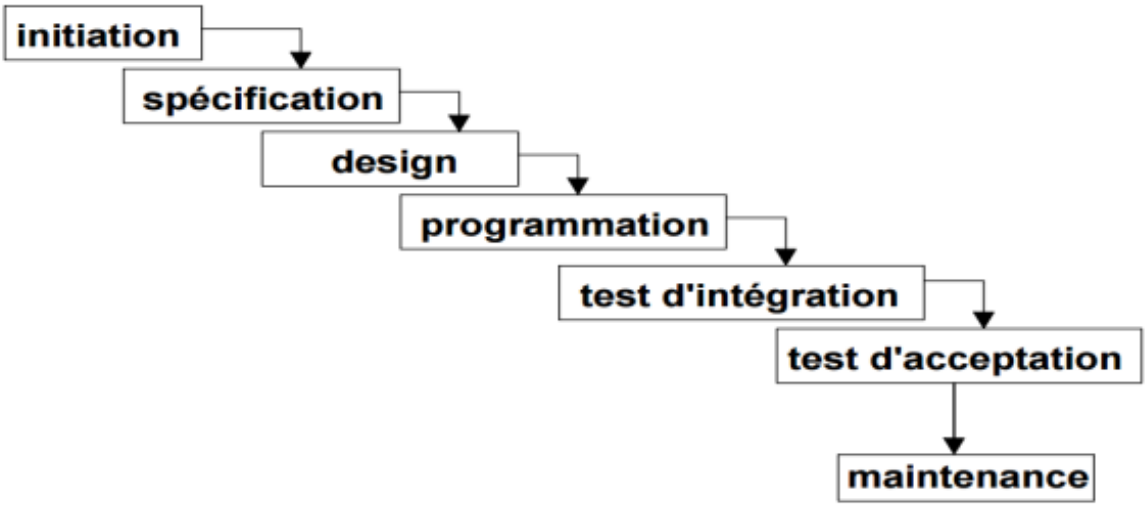

Figure I.1 Modèle de développement en cascade

I-3.1 Modèle en cascade du processus de développement d'un système embarqué

La contrainte d'un développement linéaire du modèle précédent à conduit au modèle dit en "V" (Figure I.2). Il constitue une autre manière de présenter la démarche linéaire. C'est donc une amélioration de la forme en cascade. L'intérêt réside dans de liens entre les étapes produits et celles de validation et vérification. Le parcours du V se fait de gauche à droite en suivant la forme de la lettre V; les activités de construction précèdent les activités de validation et vérification avec cette particularité que ces dernières sont préparées dès la phase de construction (flèches de gauche à droite).

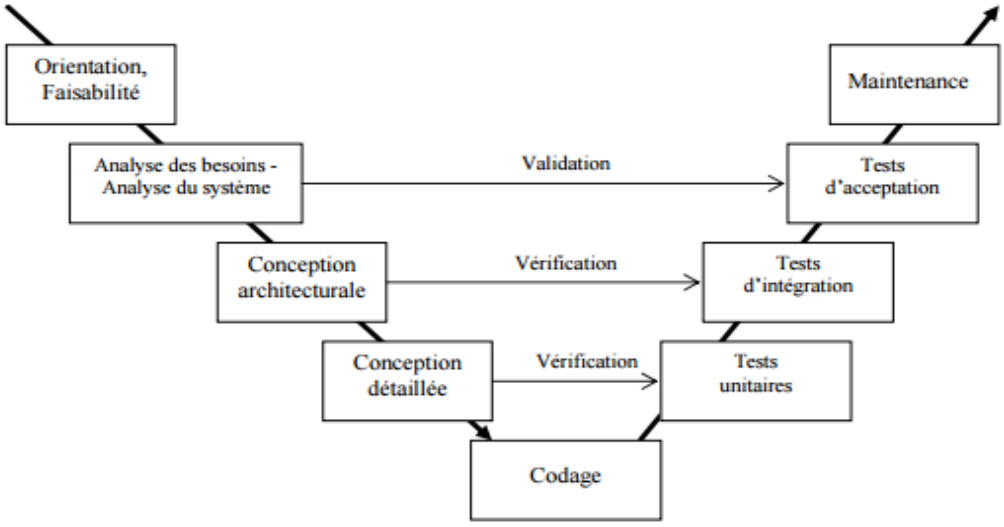

Figure I.2 Modèle en "V"

I-3.2 Le modèle en spirale

Le modèle en spirale (Figure I.3) favorise une conception itérative jusqu'à l'aboutissement à une conception détaillée qui est implémentée en suivant un cycle de développement en cascade classique. Chaque itération passe par quatre phases : définition des objectifs, identification des risques, développement et test puis préparation de la prochaine itération.

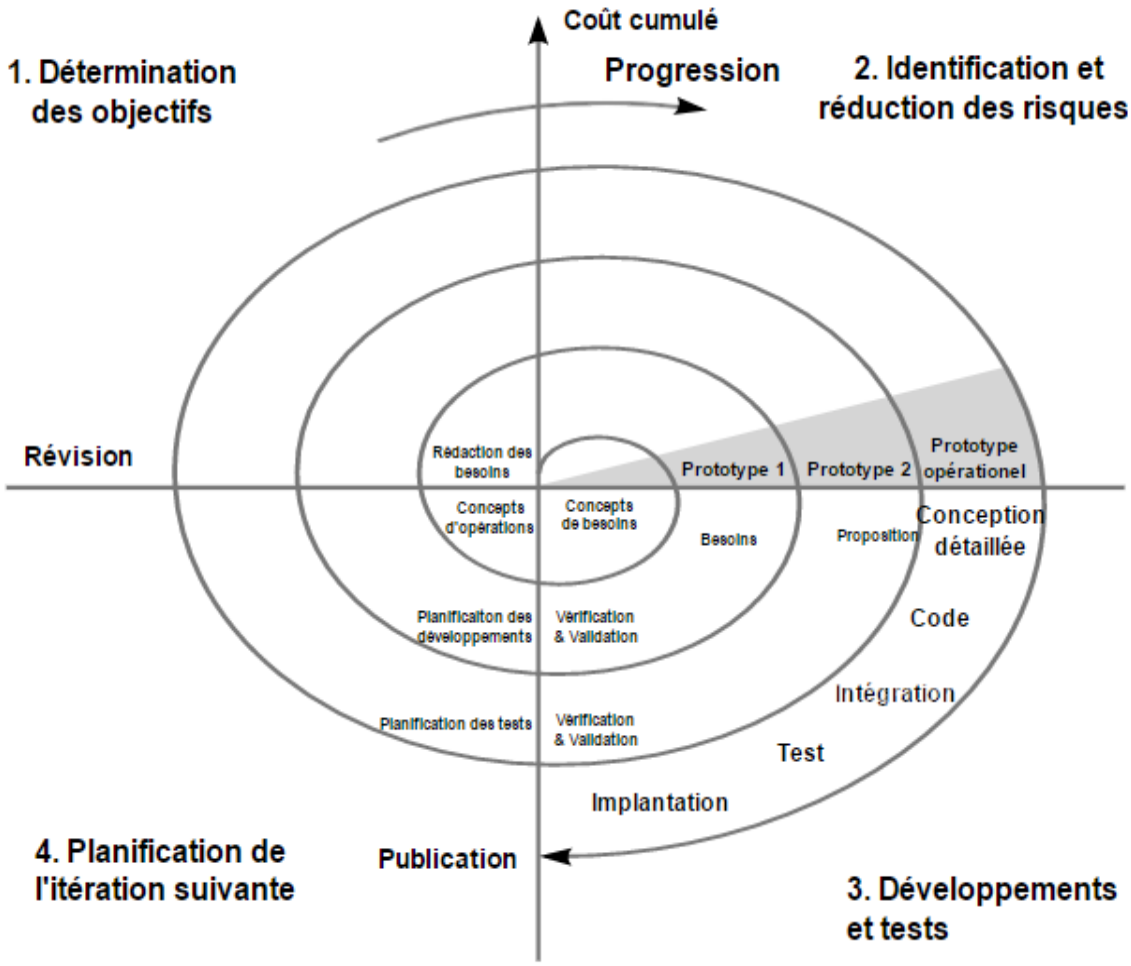

Figure I.3 Modèle en Spirale

I.3 Méthodes de développement

Les méthodes de développement complètent les cycles de vie de développement d'une application embarquée.

I-3.1 Développement Basé-Composants (Components Based Design - CBD)

L'adoption de l'approche de développement basé composants (Component-Based Développement CBD) est naturelle pour la conception des systèmes embarqués basés-COTS.

Le CBD est une approche qui stipule que le système doit être subdivisé en composants, réaliser ces composants puis les rassembler [3]. Les composants sont choisis et adaptés au lieu d'être réalisés localement. Une veille technologique continue du « marché » des COTS est, donc, nécessaire afin d'assurer un développement efficace.

I-3.2 Développement Itératif et Incrémental (Iterative Incremental Development - IID)

La combinaison de l'approche itérative et incrémentale implique l'implémentation du système en rajoutant de petits composants/fonctionnalités progressivement (incrément) et en améliorant le système par cycles répétés (itération). L'amélioration du système sous-entend la définition de métriques qui quantifient les performances du système par rapport aux spécifications fixées dans le cahier des charges.

I-3.3 Développement Dirigé par le Test (Test Driven-Development - TDD)

Les contraintes relatives aux systèmes embarqués imposent un processus basé sur les essais [3]. Le processus de conception de tels systèmes ne doit pas se contenter de la vérification et la validation du système une fois que ce dernier est implémenté mais les tests doivent êtres une partie préalable au processus de conception. On parle alors d'un développement piloté par le test (Test-Driven Development TDD). Ces tests doivent permettre la mesure/l'estimation des métriques qui quantifient le degré de concordance des performances du système par rapport aux spécifications.

I-3.4 Développement Basé Modèle (Model Based-Development - MBD)

Les moyens de simulation sont devenus de plus en plus performants permettant le déploiement de simulations de plus en plus complexes, grâce à l'accroissement de la puissance de calcul des plateformes matérielles, avec un temps d'implémentation de la simulation très

Réduit grâce au développement des logiciels de simulation. Ceci a permis l'émergence de l'approche basée-modèle qui consiste à concevoir et tester les systèmes en se basant sur

Des modèles simulés dans un environnement virtuel.

Cette approche est très répandue dans le domaine du développement des systèmes

Embarqués dédiés à la régulation. Exploite le modèle de cycle de développement (Figure I.5)

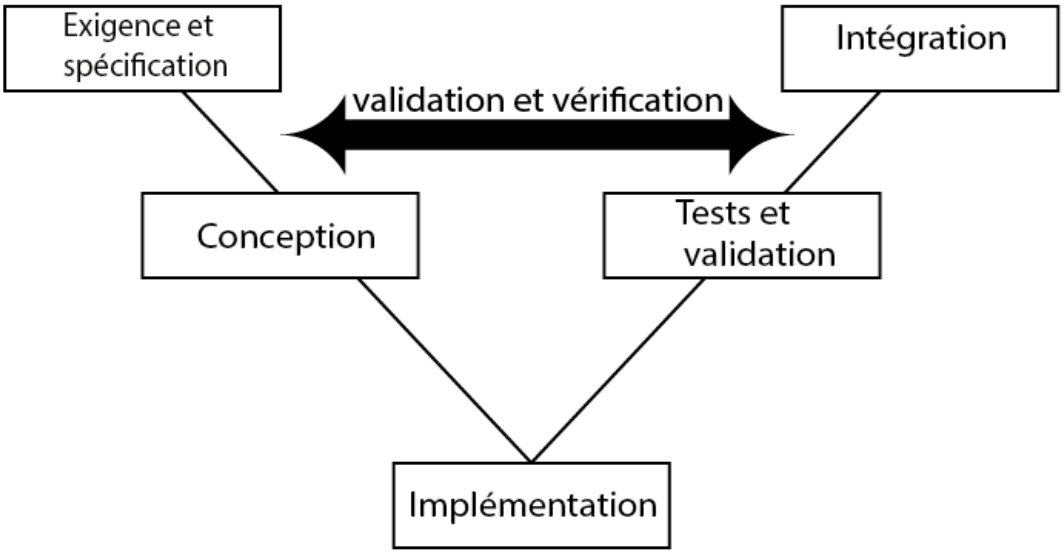

Figure I.5 Modèle en "V" appliqué au Modèle Based Design

La phase exigence et spécification traduit les exigences sous forme d'une liste de spécifications conduisant à une traçabilité entre l'exigence et le modèle.

La phase de conception englobe la modélisation sous forme de bloc ou de machine d'état. La validation se fait par les simulations.

La phase implémentations constitue le point fort de Model Based Design à travers la possibilité de génération de code automatique soit en description de bas niveau (HDL) ou haut niveau tel que c/c++.

La phase de test et de validation peut se décliner sous diverses formes :

- MIL ou Model in the loop. Le modèle décrivant le comportement du système cible est valide par simulation sans génération de code (bas ou haut niveau). Le modèle est couplé à un modèle de l'environnement (c'est à dire sans existence réelle).

- Software in the-loop (SIL). Cette forme consiste en la génération d'un code pouvant être compilé et exécuté sur l'ordinateur hôte. Le modèle de l'environnement reste virtuel (modélisé dans l'environnement de développement).

- Processor in the-loop (PIL). Cette forme est similaire à la forme précédente sauf que le code est généré et exécuté sur le processeur réel. L'environnement reste simulé.

- Hardware in the loop (HIL). Peut être vu comme la combinaison de la forme précédente avec cependant un environnement prenant sa forme réelle.

1-3.4.1 Avantages de Model-Based Design

- Les spécifications sont exécutables
- Génération automatique de code possible.
- Traçabilité entre : Exigence, modèle, code
- Intégrité des tests pendant toutes les phases de conception pour identifier et corriger les erreurs

I-6 Conclusion

A travers ce chapitre, nous avons contextualisé la méthode de développement que nous aurons à choisir. Il nous parait clair que sachant le travail antérieur auquel nous somme rattaché, la méthode la plus adaptée sera celle qui pourra capitaliser les développements antérieur tout en garantissant les spécifications initiales.

Ceci se fera après une présentation des développements liés au projet Moto IFSTAR.

Chapitre II : Les simulateurs 2RM

II-1 Introduction

Un simulateur est un dispositif physique et/ou informatique dont l'objet est de reproduire le plus fidèlement possible le comportement d'un système de référence réel.
On peut distinguer deux catégories de simulateurs : les simulateurs sans mouvement et les simulateurs à base mobile. Une plateforme mobile peut significativement améliorer le réalisme de la simulation de conduite. Nous intéresserons particulièrement aux simulateurs des deux roues motorisées (2RM).

Pour donner une base de compréhension suffisante nécessaire à la contextualisation de notre travail, nous donnons dans ce qui suit une vue générale des simulateurs 2RM.

II-2 Aux origines de l'étude des simulateurs 2RM

L'exploitation d'architectures mécaniques pilotées par des électroniques adaptées conjuguées à des techniques de simulation et d'animation graphique, communément appelés environnements virtuels, a rendu possible le développement simulateurs de conduite.

Ces derniers semblent être une alternative intéressante pour répondre à plusieurs défis tels que l'apprentissage de la conduite, la sensibilisation aux risques, la sécurité routière, etc.

Les conducteurs de deux-roues motorisés sont considérés comme les usagers de la route les plus vulnérables (statistiques) [1]. Malgré les mesures préventives aussi bien que répressives mises en place, le nombre de tués chez les motocyclistes n'a pas baissé alors que de nets progrès ont été constatés chez les conducteurs de voitures. Le risque d'être tué dans un accident reste en effet 19 [1] fois plus élevé pour un motocycliste que pour un conducteur de voiture de tourisme.

Plusieurs accidents sont la cause directe d'une méconnaissance des conducteurs de deux-roues motorisés des risques encourus dans certaines situations de conduite et des conséquences dramatiques qu'elles peuvent provoquer [1].

Aussi, les simulateurs automobiles et motos ont été développés, comme pour les avions, en tant qu'outil de formation, mais ils servent également pour l'étude du comportement humain, voire même en tant que dispositif de prototypage pour les concepteurs; ce qui leur permet de tester de nouvelles fonctionnalités technologiques. On peut dire que ces simulateurs peuvent s'inscrivent dans un contexte beaucoup plus larges qui est celui des jeux utiles (seriousgames).

Dans ce contexte, de nombreux simulateurs de conduite ont été élaborés au cours de ces dernières décennies [1] et ce, pour différents types de véhicules. On peut distinguer deux catégories de simulateurs : les simulateurs sans mouvement et les simulateurs à base mobile.

Les premiers sont uniquement construits autour d'un écran assurant un retour visuel tandis que les derniers produisent, en plus de l'information visuelle, des mouvements cohérents avec ceux d'un véhicule réel. Souvent, une plateforme mobile peut significativement améliorer le réalisme de la simulation de conduite.

II-3 Composantes d'un simulateur

Tout simulateur peut être considéré comme prototype car il n'existe aucune norme de construction associée. Sa conception est le fruit d'une collaboration pluridisciplinaire.

Un simulateur de conduite est le résultat de la composition de plusieurs sous-systèmes (Figure I.1) devant fonctionner de façon synchrone pour créer un environnement de conduite proche du véhicule réel. Ces sous-systèmes doivent être synchronisés avec un minimum de retard entre l'action du conducteur et la réaction du simulateur.

De façon générale, les composants d'un simulateur de conduite se répartissent en deux catégories :

a- La partie physique (ou visible)

Elle contient l'ensemble des composants avec lesquels le conducteur interagit et, qui influent sur son comportement. On y distingue :

- Le cockpit : un intérieur semblable sinon identique à celui d'un véhicule réel incluant les commandes (guidon, pédales, freins, ...).

- Le système visuel permettant la projection de paysages simulés d'environnements réels dans lesquels le conducteur sera immergé.

- La plateforme (ou le mécanisme de mouvement) qui donne au cockpit son mouvement dans l'espace.

- Le système sonore qui reproduit l'environnement sonore de la conduite : bruit du moteur,
klaxons, crash, vent, grincement des roues,...

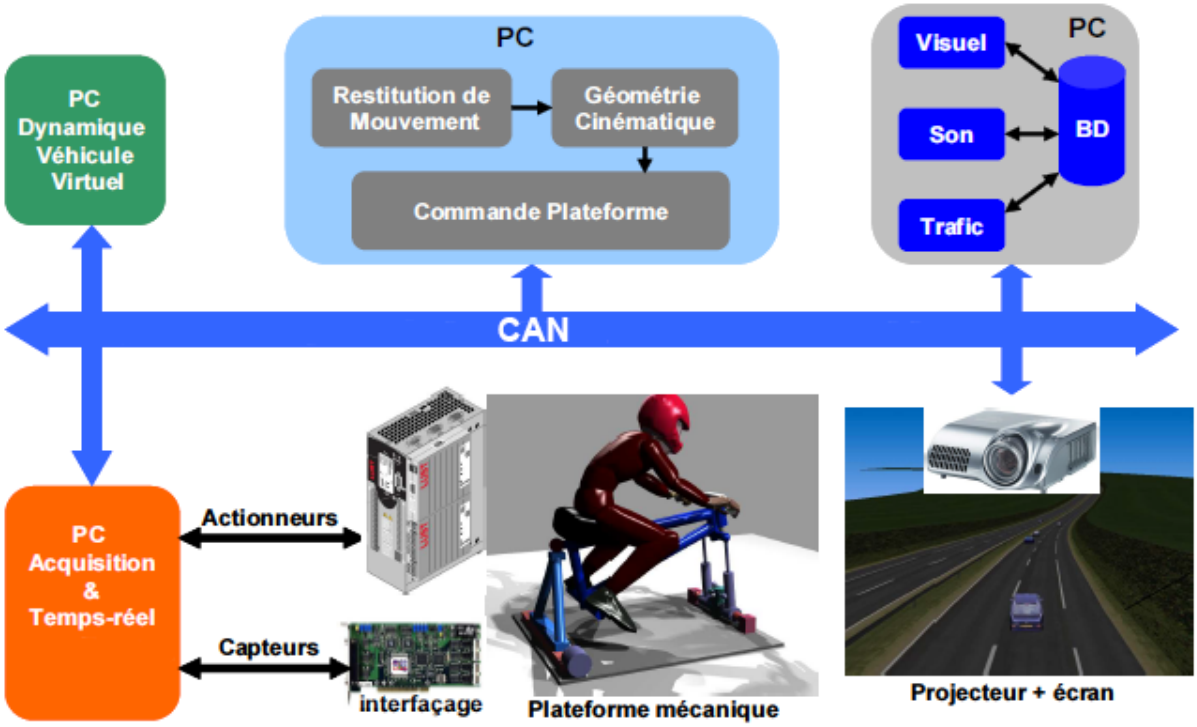

Figure I.1 : Exemple de composantes d'un simulateur [1]

b- La partie embarquée

Englobe tous les composants enfouis dans l'environnement de conduite réel. On y distingue :

- L'électronique dédiée à l'instrumentation incluant les divers capteurs et les interfaces d'acquisitions aussi bien digitales qu'analogiques
- Les calculateurs dédiées soit à l'implémentation des modèles de calcul (2RM et Plateforme), soit au pilotage de l'instrumentation
- Les divers systèmes d'interconnexion (bus, liaison point à point, ...)

II-4 Classification des architectures des simulateurs de conduite

Les simulateurs s'articulent essentiellement sur l'architecture de la plateforme; l'objectif étant d'offrir des ressentis les plus proches des conditions réelles. Diverses architectures existent et sont généralement classées suivant leurs types. Chaque architecture vise à atteindre des objectifs précis. On en distingue 4 types de plateformes déduites des modèles de la robotique.

II-4-1 Plateformes à base fixe

Ces plateformes ne possèdent aucun mouvement mécanique (Figure I.2). Les effets Inertiels et d'autres effets dynamiques sont inexistants. Aucune fonction de
De restitution de mouvement n'y est intégrée. Les ressentis sont essentiellement induits Par le retour visuel.
Dotées de système de reproduction sonore, elles se composent d'une cabine instrumentée de mécanismes de retour d'effort sur le volant et de siège vibrant permettant la création d'un environnement de conduite tentant de mimer les conditions réelles.

Figure I.2 : Exemple de plateforme à base fixe [1]

II-4-2 Plateformes à structure série

Dans cette configuration, la cabine est portée par une structure mécanique constituée
D'une mise en série d'articulations reliée à une base fixe (Figure I.3). Le modèle est illustré par la figure I.3. La figure I.4 illustre un exemple de plateforme à structure série. Les performances dynamiques se dégradent [1] en présence d'une charge importante et pour des vitesses élevées

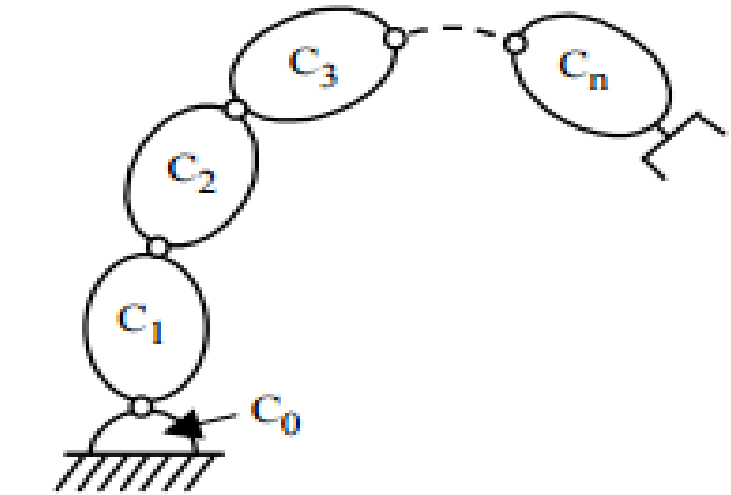

Figure I.3 : Modèle de la plateforme série [1]

Figure I.4 : Exemple de plateforme à structure série [1]

II-4-3 Plateformes à structure parallèle

Elles sont constituées de chaînes cinématiques fermées (Figure I.5). L'organe terminal portant la cabine est supporté par plusieurs actionneurs [1]. Un exemple est illustré par la figure I.6.

On peut distinguer plusieurs avantages par rapport à la plateforme série :

- Les charges peuvent être très importantes
- Plus grande précision (plus faible couplage)
- Meilleure rigidité
- Une dynamique plus riche offrant des couples importants à des vitesses de Fonctionnement élevées.

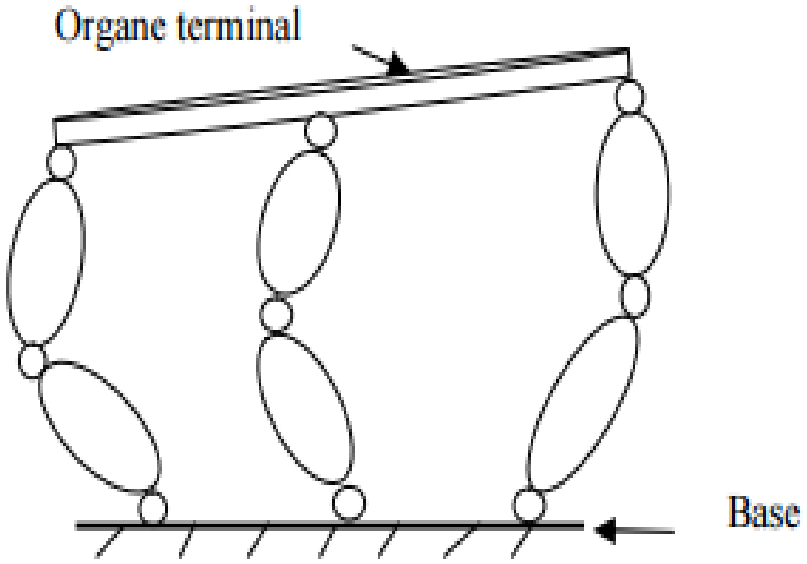

Figure I.5 : Modèle de la plateforme parallèle [1]

Figure I.6 : Exemple de plateforme à structure série [1]. Université de Suède

II-4-4 Plateformes à structure hybride

Le modèle multi-corps est donné par la figure I.7. L'objectif visé étant de combiner les avantages des deux précédentes architectures à savoir série et parallèle.

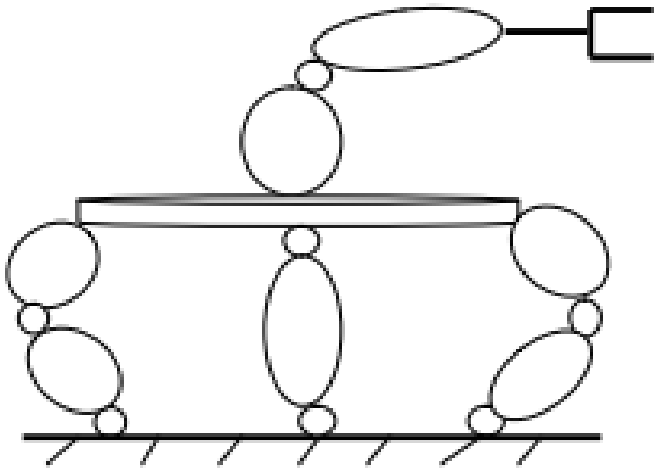

Figure I.7 : Modèle de la plateforme hybride [1]

Figure I.8 : Exemple de plateforme à structure hybride [1]

II-5 Exemples de simulateurs de véhicule deux-roues

Les deux-roues n'ont fait, depuis longtemps, l'objet que d'une priorité secondaire alors que dans le domaine de l'accidentologie, le nombre de tués par rapport aux quatre roues motorisées est important.

La bibliographie des simulateurs de véhicule deux-roues est pauvre comparée à celle des véhicules automobiles. Par leur histoire, les principaux travaux ont été effectués par des institutions industrielles japonaises et italiennes.

Nous donnons ci-après une brève présentation de quelques simulateurs 2RM

II-5.1 Simulateurs Honda

Les deux objectifs du développement d'un simulateur moto que s'est assigné le constructeur Honda se résume à :

- Etudier la manœuvrabilité d'une 2RM sur un simulateur de conduite.
- Etudier la faisabilité de la formation pour assurer la sécurité des sujets.

Deux prototypes ont vue le jour dès 1988 (ce qui parait relativement récent); le 2ème (Figure I.8) étant l'amélioration du premier.

Le but recherché est de tester la faisabilité de la simulation de conduite à reproduire des manœuvres de base pour la formation des motards.
Cette plateforme, de type parallèle, présente 5 degrés de liberté (roulis, lacet, tangage, latéral et rotation du guidon).

Figure I.8 : Vue du simulateur 2RM Honda (2ème Prototype)

Figure I.9 : Honda Riding Trainer

Honda commercialise son simulateur “low-cost” (bas coût) de conduite deux-roues (Figure I.9) [1]. C'est un modèle à base fixe. Le conducteur écoute au préalable les recommandations de l’ordinateur. Il s'engage sur un parcours qu'il aura choisi et circule à travers un environnement virtuel. Il reçoit un bilan établi par l'application qui pointe les erreurs de conduite et commente les incidents ou accidents graves.

On voit un des usages des plus pertinents des simulateurs 2RM dans le contexte de son usage par les motos écoles et par les formateurs professionnels.

II-5.2 Simulateur Moris

La figure I.10 montre une vue du simulateur de Moris. Il est est né d’une collaboration entre le laboratoire PERCRO (Italie) et le fabricant de véhicules deux-roues Piaggio.
Implémente une plateforme parallèle type Gough-Stewart à 6DDL hydrauliquement actionnés. Un châssis original de scooter réel, Piaggio 175 a été fixé sur la plateforme mobile. Les principales commandes ont été instrumentées et un moteur a été adjoint afin de reproduire les vibrations du moteur.

Figure I.10 Simulateur MORIS du laboratoire PERCRO - Italie

II-5.3 Simulateur IFSTAR

C'est le simulateur (Figure I.11) dont l'électronique embarquée sera l'objet de notre étude. La moto est positionnée sur trois vérins. Deux vérins remplacent la fourche

permettant de modifier l'assiette de la moto. Un troisième vérin, monté sur une rotule fixée sur un rail, restitue les informations de lacet. Un quatrième vérin, monté sous forme longitudinale, reproduit l'effet de la variation de la vitesse.

L'instrumentation englobe plusieurs capteurs (pressions, freins, comodo, etc ...) piloté par un calculateur embarqué. La communication entre les divers organes électroniques est assurée par bus CAN. Un ensemble d'ordinateurs de table assurent l'exécution des modèles moto et celui de la plateforme ainsi que l'exécution du modèle de l'environnement virtuel.

Elle se conduit en mode contre-braquage, proche de la conduite réelle. Le guidon est piloté par un moteur pour le ressenti du retour d'effort. Elle a été conçue pour l'étude du comportement de conducteurs 2RM.

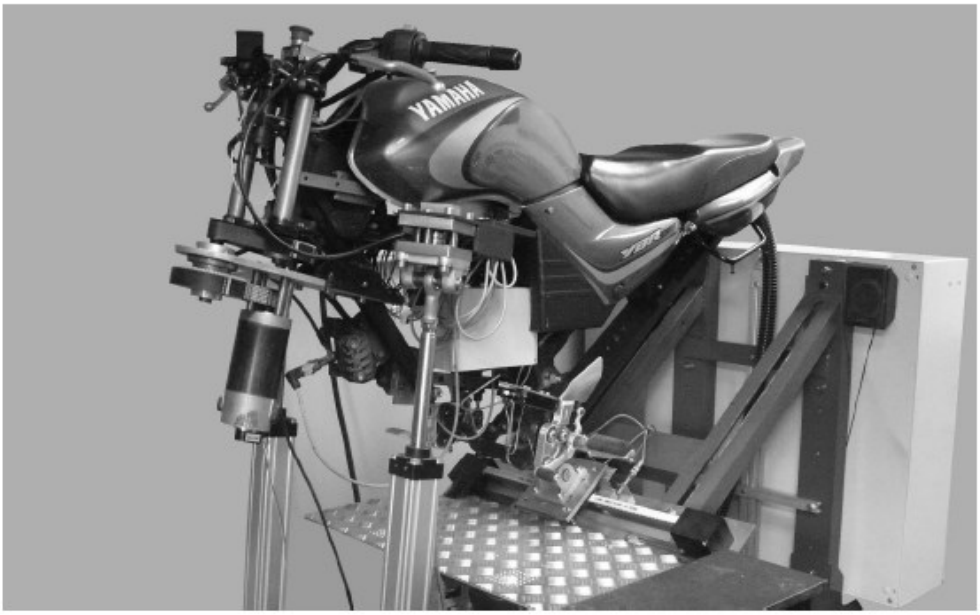

Figure I.11 Simulateur IFSTAR

II-6 Conclusion

Nous avons présenté dans ce chapitre les éléments nécessaires à la contextualisation de notre projet. C'est ainsi que nous avons introduit la plateforme IFSTAR à travers les diverses plateformes existantes et déduites de la problématique de l'usage des véhicules 2RM.

La plateforme IFSTAR, telle que présentée, est une plateforme de type parallèle. Sa conception et sa réalisation remonte à une dizaine d'année. La nécessité de la faire évoluer a conduit à la refonte de son électronique embarquée avec comme contrainte son

adaptabilité à toute plateforme de simulation possible tout en utilisant des plateformes électroniques embarquées bas coût.

L'ensemble de son étude a été faite sur des environnements de développement non homogènes avec cette particularité, comme on l'a précisé précédemment, que des ordinateurs de table implémentent les deux modèles (plateforme et moto). Ces implémentations ont été faites sous environnement de développement MATHWORKS (MATLAB/SIMULINK) y compris la partie temps réel.

Note étude consistera à migrer ces deux implémentations tournant sur des PC de tables vers des cibles embarquées bas coût, soit via un codage bas niveau ou à travers une approche haut niveau. Ce sera le but et le développement de nos prochains chapitres.

Chapitre III : Etude de l'architecture du simulateur deux-roues d'IFSTAR

III-1 Introduction

Le but qu'on nous a assigné consiste en la migration des modèles implémentés sur des PCs de tables vers des cibles embarquées. L'étude de l'existant devient une exigence préalable pour pouvoir choisir et adapter une solution d'implémentation donnée. Ce sera l'objet de ce chapitre.

Comme on la précisé, les modèles ont été implémentés sur des environnements de développement de haut niveau pilotant des cartes d'interface spécialisées.

III-2 Description de la boucle de simulation

La figure III-1 décrit l'architecture du simulateur 2RM d'IFSTAR. On y distingue trois PCs de table; les deux premiers exécutant l'implémentation du modèle de la Moto et celui de la plateforme tandis que le dernier assure l'exécution du modèle de l'environnement virtuel.

L'intercommunication est assurée par deux liaisons distinctes à savoir UDP et CAN. Chacune de ces dernières sert à transmettre des trames véhiculant les entrées et les sorties de chaque modèle.

La PC de table (xPC Target) constitue le nœud par lequel passent les données capteurs et l'exécution du modèle de la plateforme. L'exécution de ce dernier assurera la commande adaptée des vérins.

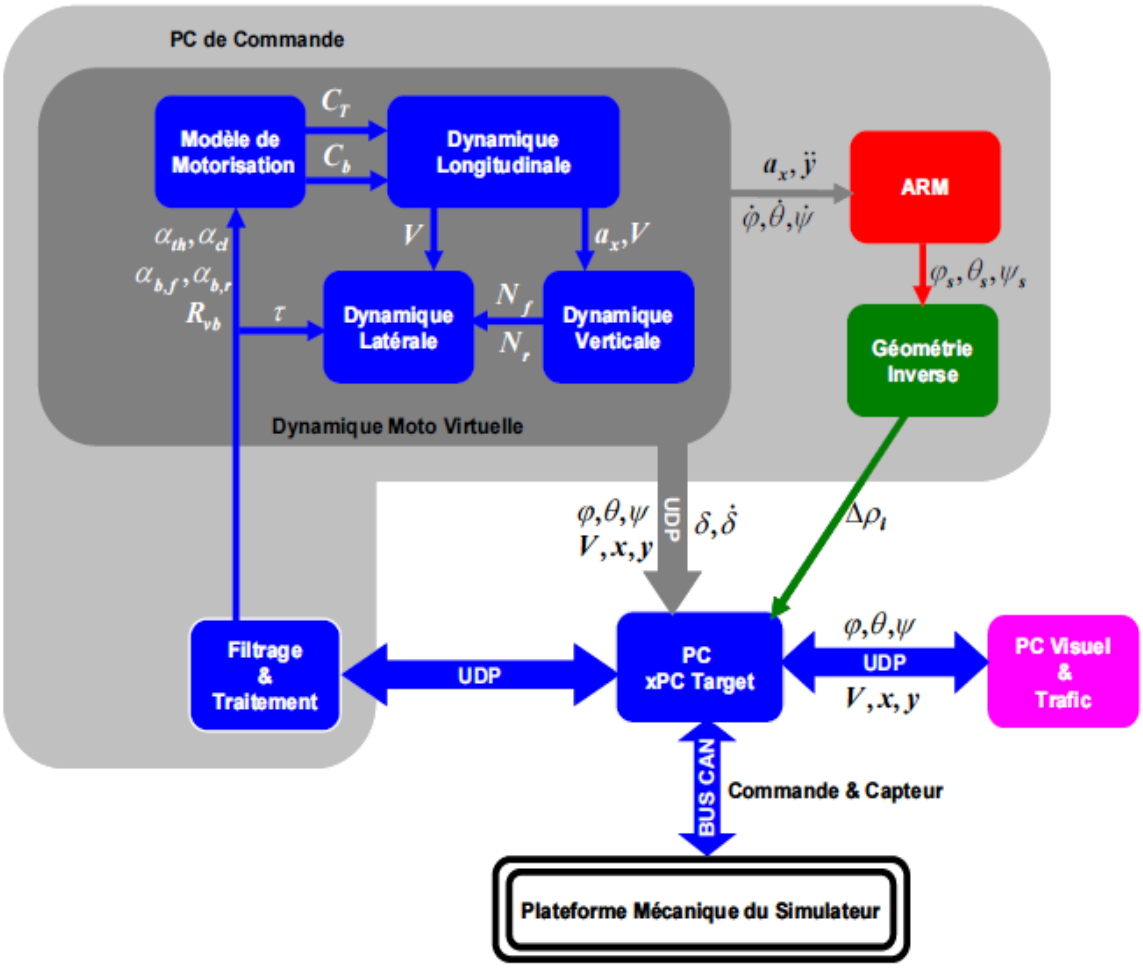

Figure III-1 : Architecture du Simulateur 2RM - IFSTAR

III-2.1 Visuel / Trafic

Comme indiqué précédemment, la gestion du rendu visuel et celle du trafic est assurée par un PC table.
Le module visuel, développé par l'INRETS/MSIS, est basé sur l'utilisation de la librairie SGI Performer.

Le modèle de trafic (résultat du projet ARCHISIM [1]) a pour finalité la mise en œuvre d'une simulation réaliste de situations routières assurant une interactivité avec la scène visuelle.
Par concept de trafic est ciblé le déploiement d'un modèle décrivant les actions individuelles en interactions avec des différents acteurs apparaissant dans le rendu visuel.

Ces deux modules sortent du cadre de notre étude.

III-2.2 Dynamique de la moto virtuelle

Le mouvement de la plateforme d'un simulateur de conduite est défini à partir d'un modèle qui décrit la dynamique du véhicule virtuel déduit d'une modélisation préalable. Ce dernier permet de calculer les états du véhicule en réponse aux diverses actions du conducteur.

III-3 Modèle des deux roues utilisées

Ce modèle dynamique servira à la visualisation du comportement d'une deux roues sur le visuel. Il dérive des études générales de la dynamique des véhicules deux roues. L'état de l'art dans ce contexte montre que l'étude des deux-roues semble être plus ancienne que celle de l'automobile. La première étude portant sur la stabilité et l'équilibre d'un véhicule à deux roues date de 1899 (Whipple) [1].
Les deux modèles les plus répondus sont respectivement celui de Sharp [1] et celui dit de Meijaard [1]. Ces deux modèles ont été implémentés sur le simulateur d'IFSTAR. Il nous a été spécifié l'exploitation du modèle de Meijaard.

III-3.1 Modèle Meijaard

Dans ces travaux, MEIJAARD a utilisé un modèle bicyclette composé de quatre corps rigides (Figure III-1) : la roue arrière « R », le solide arrière (rear frame) « B » avec le corps du pilote qui est considéré comme rigidement lié à ce cadre, le solide avant « H » (front frame) comprenant le guidon et la fourche et enfin la roue avant « F ». Ces corps sont reliés entre eux par trois liaisons pivots (pivot arrière, axe de direction, pivot avant). Tous les corps sont supposés être symétriques par rapport au plan médian de la bicyclette. Les roues sont supposées être infiniment minces sans glissement dans la direction longitudinale et latérale.

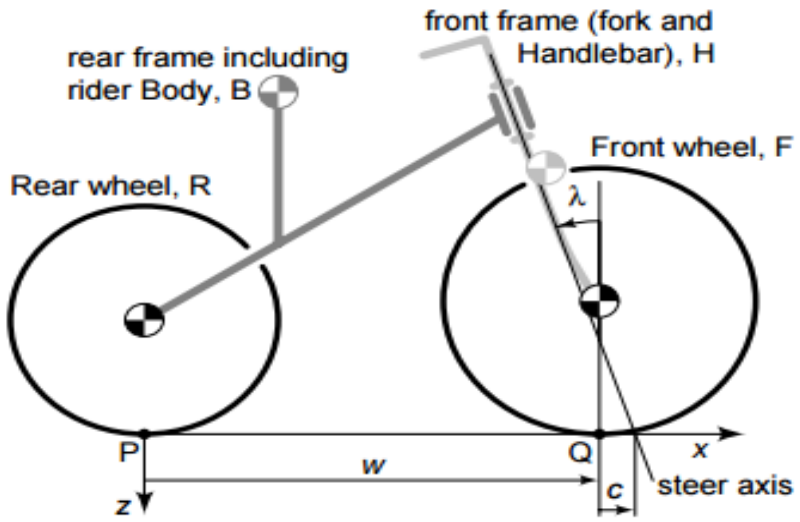

Figure III.2 modèle bicyclette

L'équation de mouvement qui en est déduite [2] est donnée par :

$$M * \begin{pmatrix} \ddot{\phi} \\ \ddot{\delta} \end{pmatrix} + v * C_1 * \begin{pmatrix} \dot{\phi} \\ \dot{\delta} \end{pmatrix} + [g * K_0 + v^2 * K_2] * \begin{pmatrix} \phi \\ \delta \end{pmatrix} = \begin{pmatrix} T_\phi \\ T_\delta \end{pmatrix}$$

Avec :

- M, C_1, K_0 et K_2des matrices (4x4) dépendant uniquement de la géométrie de la moto
- $v$la vitesse de la moto
- $g$la constante de gravité (9.81 m/s²)
- ϕl'angle de roulis de la moto
- δl'angle du guidon de la moto
- $T_\phi$le couple de roulis exercé par le conducteur
- $T_\delta$le couple au guidon exercé par le conducteur

Cette équation peut également s'écrire sous la forme :

$$\begin{pmatrix} \ddot{\phi} \\ \ddot{\delta} \end{pmatrix} = A * \begin{pmatrix} \dot{\phi} \\ \dot{\delta} \end{pmatrix} + B * \begin{pmatrix} \phi \\ \delta \end{pmatrix} + F$$

Avec (si on noteM^{-1}la matrice inverse de$M$) :

- $A = -v * M^{-1} * C_1$
- $B = -M^{-1} * [g * K_0 + v^2 * K_2]$
- $F = M^{-1} * \begin{pmatrix} T_\phi \\ T_\delta \end{pmatrix}$

On voit, à travers ce modèle, que les deux effets induits par le conducteur (vitesse longitudinale et couple de roulis) conditionnent les angles d'Euler :

- Rotation autour de l'axe des X d'un angle φ (roulis)
- Rotation autour de l'axe des Y d'un angle θ (tangage)
- Rotation autour de l'axe Z d'un angle ψ (lacet)

L'équation de l'angle de lacet ψ peut être déduite [2] par :

$$\dot{\psi} = \frac{(v\partial + c\dot{\partial})}{w} \cos \lambda$$

Le point de contact arrière est donné par [2]:

$$\dot{x}_p = v \cos \psi$$
$$\dot{y}_p = v \sin \psi$$

L'intégration des deux équations précédentes permet de reconstruire la trajectoire réalisée par le véhicule deux roues par l'interpolation des points (x, y) de la roue arrière à chaque instant.

Exemple de quelques paramètres du modèle pour une moto

Wheel base w = 1.42 m

- Trail c = 0.1155 m
- Steer axis tilt λ= 27.2 degrés (angle de chasse)

Rearwheel

- Radius r_R=0.3 m
- Mass m_R=13.5 kg
- Mass moments of inertia :
  - I_{Rxx} = 0.407025 kg.m²
  - I_{Ryy} = 0.81 kg.m²

Front wheel :

- Radius r_F=0.3 m
- Mass m_F=11.5 kg
- Mass moments of inertia
  - I_{Fxx} = 0.346725 kg.m²
  - I_{Fyy} = 0.69 kg.m²

Rear body and frame assembly

- Position centre of massx_B= 0.7 m et z_B= - 0.6 m
- Mass m_B= 150 kg
- Mass moments of inertia
  - I_{Bxx} = 16.2353 kg.m²
  - I_{Bxz} = 4.2353 kg.m²
  - I_{Byy} = 19.4118 kg.m²
  - I_{Bzz} = 4.9412 kg.m²

Front Handlebar and forkassembly

- Position centre of massx_H= 1.22 m et z_H= - 0.8 m
- Mass m_H= 20 kg
- Mass moments of inertia
  - I_{Bxx} = 0.2946 kg.m²
  - I_{Bxz} = - 0.0378 kg.m²
  - I_{Byy} = 0.3 kg.m²
  - I_{Bzz} = 0.0354 kg.m²

III-4 Modèle de la Plateforme du Simulateur deux roues

La solution de l'architecture préconisée par l'équipe d'IFSTAR devait aller dans le sens de la restitution fidèle des mouvements d'une deux roues motorisée avec la contrainte de développement d'un outil bas-coût commercialisable [1].

Le résultat de l'étude effectuée a conduit à la plateforme du simulateur donnée par la figure III.3. Elle se compose d'un bâti métallique et d'une partie supérieure mobile. Le bâti sert de support pour le montage des vérins et du système glissière arrière. Un châssis de moto est fixé sur la partie supérieure constitué d'un cadre métallique. Un module construit autour d'une motorisation pilotant un mécanisme de réduction est destiné à assurer un

retour d'effort guidon (Figure III-4). L'électronique de pilotage et l'implémentation de la commande de régulation n'est pas incluse dans la problématique objet de notre étude.

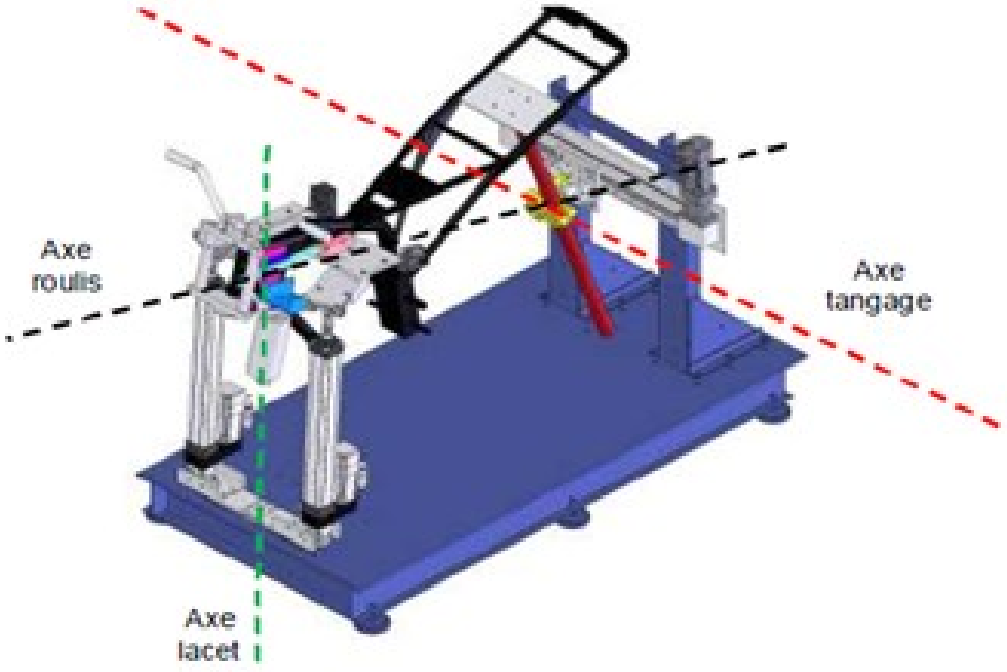

Figure III.3 Modèle CAO de la plateforme du simulateur d'IFSTAR

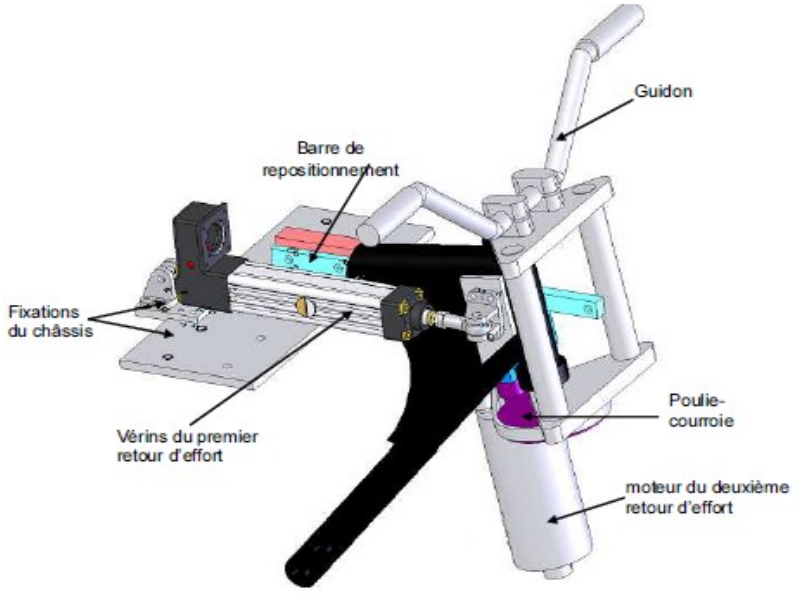

Figure III.4 Modèle CAO du système de retour d'effort guidon du simulateur d'IFSTAR

III-4.1 Modèle géométrique inverse

Ce modèle est destiné, à partir des angles d'Euler, à déduire les déplacements linéaires des vérins mis en jeu.

Le modèle cinématique correspondant à la conception donnée par la figure III.3 est représentée par la figure III.5
Les points d'attaches des deux vérins avant et du support de la glissière arrière avec le bâti de la plateforme sont représentés par les points B1, B2 et B3 respectivement. Les points d'attaches supérieurs des deux vérins avant et de la glissière arrière avec la plateforme mobile sont représentés par les points P1, P2 et P3 sont respectivement.

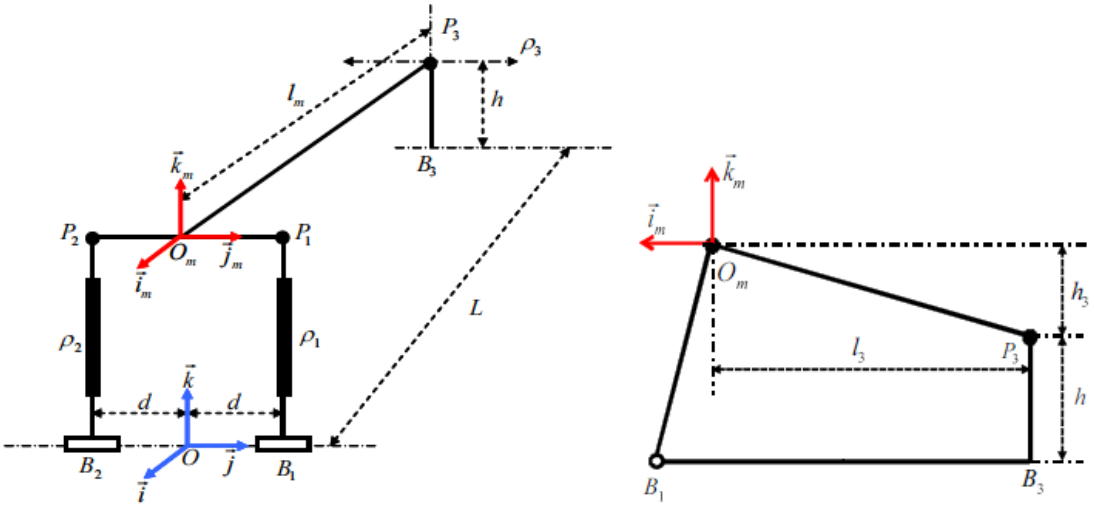

Figure III.5 Modèle cinématique de la plateforme.

Le modèle dynamique de la plateforme est donné par [1] :

$$M'\dot{V}_\rho + C' + G_g + \Phi_q^T \Lambda = \mathcal{J}_{-1}^T F$$

Avec :

$\dot{V}_\rho = [\ddot{\rho}_1 \quad , \ddot{\rho}_2 \quad , \ddot{\rho}_3]$: Le vecteur des accélérations articulaires

M' : la matrice modifiée de la matrice de masse M dépendant de l'orientation de la Plateforme

C' : le vecteur transformé du vecteur de Coriolis C dépendant de l'orientation et de la vitesse angulaire de la plateforme

Gg : le vecteur contenant les éléments gravitationnels

Λ : le vecteur des multiplicateurs de Lagrange

Φ : la matrice des contraintes

F : les forces liées aux actionneurs

Dans le cas de la plateforme d'IFSTAR, l'ensemble de ces paramètres ont été obtenus par identification [1].

Ce modèle, développé dans la thèse [1] décrit l'ensemble des efforts internes correspondant aux forces et couples nécessaires pour contraindre les mouvements relatifs définis par la mécanique de la plateforme. Il permet, à travers l'obtention des angles d'Euler et de la vitesse linéaire, de déduire les valeurs de déplacement des vérins $\rho_1, \rho_2, \rho_3, \rho_4$

III-5 Instrumentation de la plateforme

III-5.1 Capteurs installés

Le simulateur est doté des commandes classiques d'une moto. Les informations provenant des capteurs alimentant le modèle virtuel de la moto sont donnés par le tableau III.1

On y distingue :

1- La position du guidon fournie par un codeur optique représente l'angle que le pilote imprime au guidon.

2. La position de la manette de gaz est donnée par un potentiomètre linéaire actionné par un câble relié à la manette.

3. La position de la poignée d'embrayage est construite sur le même modèle que celui de la manette de gaz.

4. La puissance de freinage de la manette de frein (frein avant) est restituée grâce à un capteur de pression (monté sur le circuit hydraulique de freinage avant). La force exercée sur la poignée est convertie en signal analogique.

5. La puissance de freinage de la pédale de frein (frein arrière) est construit sur le même modèle que le dispositif de freinage avant.

6. L'information de sélection de vitesse est construite à l'aide de deux interrupteurs positionnés de façon à détecter une montée ou une descente du sélecteur de vitesse (deux bits utilisés).

Signal	Capteur	Type	Référence
Accélérateur et embrayage	Potentiomètre linéaire	Analogique	Megatron MBW100 10KW
Freins avant et	Pression	Analogique	Gems 1000BG

arrière			
Sélecteur vitesse	Interrupteur	Binaire	Omran Z-15GL2-B
Rotation guidon	Codeur optique	Binaire	HEDS-5500
Couple guidon	Jauge de déformation	Analogique	Absent
Position conducteur	Codeur optique	Binaire	Absent
Torseur cinématique de la plateforme	Centrale Inertielle	Analogique	Absent

Table III.1 – Liste des capteurs installés

III-5.2 Carte d'acquisition cabine

Cette carte doit assurer :

- L'acquisition des signaux issus des différents capteurs et les transmettre au modèle

Implémenté sur le PC de commande pour le calcul les états de la moto virtuelle.

- La commande des actionneurs des variables articulaires en mode analogique.
- Gérer le retour d'effort guidon.

Sa constitution peut être résumée à :

- une carte secondaire sert à l'adaptation des signaux issus des capteurs.
- un microcontrôleur de la famille V853 de marque NEC dédiée à la gestion bas-niveau des

Capteurs instrumentant la moto du simulateur. On y distingue :

- 12 entrées analogiques ±10V
- 16 sorties analogiques 0-5V
- plusieurs entrées/sorties numériques
- 24 entrées binaires bufférisées dédiées à l'acquisition des signaux des deux canaux A et

B pilote des codeurs optiques

- des sorties PWM (Pulse Wave Modulation) permettant l'obtention de niveaux analogiques commandés en fréquence.
- un contrôleur intégré assurant la communication CAN (Controller Area Network)

III-5.3 Pilotage des vérins et du guidon

Les vérins assurant les mouvements de la plateforme du simulateur sont actionnés électriquement via (moteurs) via des variateurs de la famille "Lust CDD3000". (Figure III.2)

Ils peuvent assurer des commandes en :

- couple (signal analogique ±10V)
- position, avec une régulation de position externe (±10V)
- position, avec des consignes et des mesures transmises via le bus CAN

- vitesse, avec des mesures et des consignes transmises via le bus CAN

Le guidon est piloté par un moteur monté sur des réducteurs mécaniques. Le retour d'effort guidon est assuré par la commande du dit moteur par une carte de puissance. Elle fournit le courant nécessaire, à partir des signaux PWM envoyée par la carte NEC, pour produire un couple moteur donné.

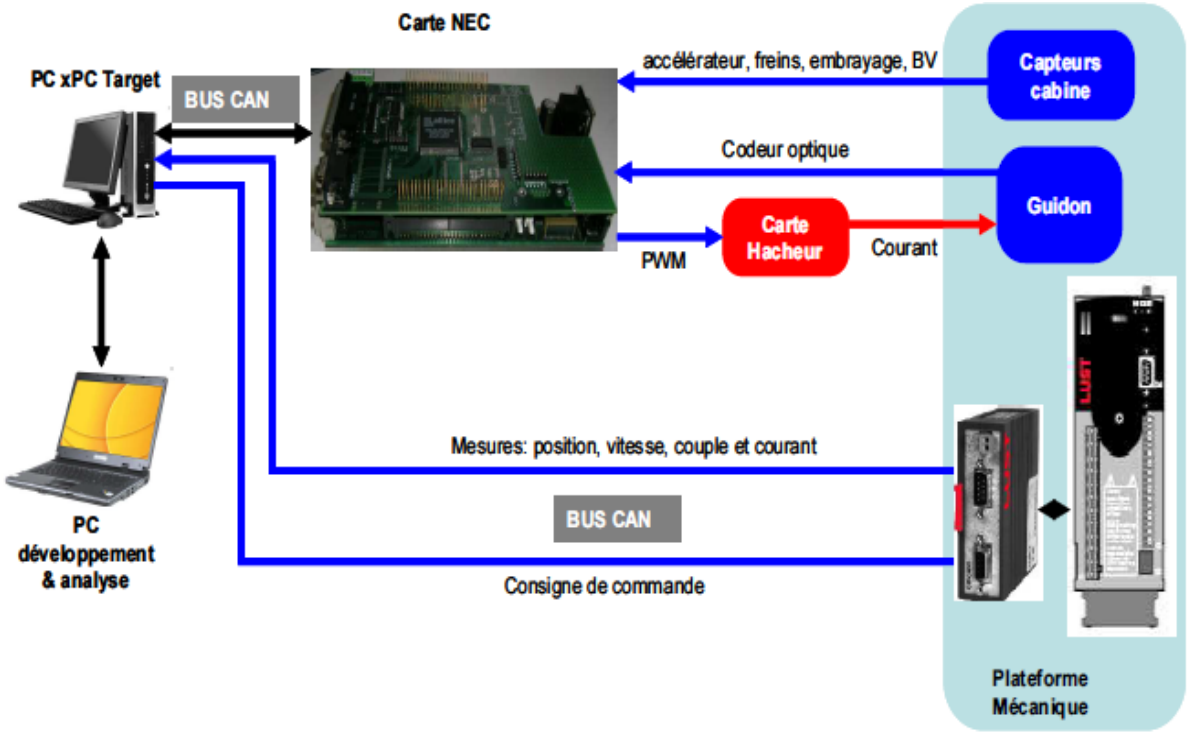

Figure III.6 : Instrumentation de la plateforme

III-5.4 Acquisition et gestion temps-réel

Simulink utilisant un vecteur de temps séquentiel pour évaluer les différents états et sorties du modèle implémenté non connecté à l'horloge du système, les sorties sont déterminées en mode du temps non-réel dépendant de la performance du PC utilisé. Ceci a conduit les concepteurs du système, pour la synchronisation et la gestion des divers états, à l'introduction d'un noyau temps-réel; en l'occurrence le gestionnaire temps-réel sous DOS (xPC Target) fournit par la boite à outil "xPC Target" (Mathworks).

III-6 Séquencement et synchronisation

La machine d'état, décrivant les séquences de synchronisation du simulateur d'IFSTAR, est donnée par la figure III.7

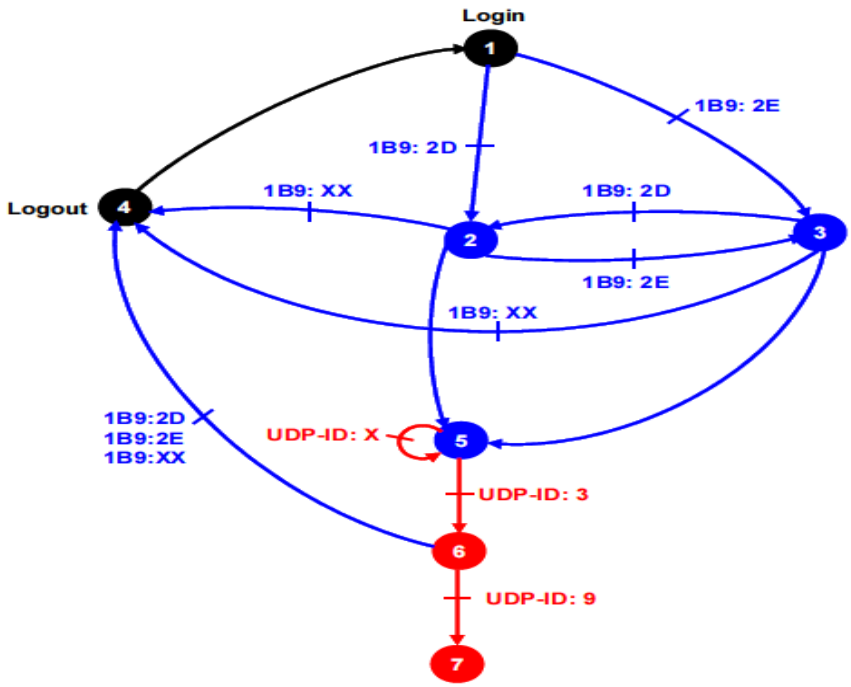

Figure III.7 les séquences de synchronisation du simulateur d'IFSTAR

1. Au début de la simulation, les actionneurs sont mis à l'état 1 "Login". Une trame CAN, définie par son identificateur, est envoyée à chaque variateur pour le positionner en mode régulation

2. La deuxième étape consiste à positionner la plateforme mobile du simulateur à la Position neutre (Vérins avant à mi-élongation et glissière arrière dans le plan vertical De symétrie de la plateforme). Ce positionnement est réalisé de manière simple, En scrutant la plage de déplacement disponible de chaque variable articulaire. En Étant en fin de course, minimale ou maximale, chaque variateur renvoi une trame D'erreur d'identificateur 1B9.

3. Etant à l'une ou à l'autre des fins de course (état 2 ou 3), l'excursion maximale de Chaque actionneur est scruté afin de déterminer le point milieu pour positionner La plateforme (état 5). A cet état, une trame CAN est envoyée en mode écriture Pour forcer la valeur de la position à zéro.

4. Une fois la plateforme mise au neutre, une requête UDP avec un ID : 1 est envoyée Pour établir une connexion au PC visuel. Si ce dernier répond avec un ID : 3, la Connexion est établie (état 6), sinon, une deuxième requête est renvoyée. A partir De cet instant, nous pouvons commencer la simulation de la conduite.

5. Si une trame d'erreur d'identificateur 1B9 : XX est détectée, les variateurs sont mis Hors régulation. De plus, une requête UDP d'ID : 9 permet de se déconnecter du visuel.

III-7 Conclusion

L'objet de ce chapitre consiste en la présentation du simulateur moto d'IFSTAR. Il nous est apparu nécessaire, malgré la difficulté de la synthèse d'un travail de thèse [1] en un chapitre, de présenter l'essentiel d'une manière sommaire pour comprendre les modèles Simulink développés.

L'objectif qui nous a été assigné réside dans la transposition de ces modèles sur des cibles embraqués.

Un aperçu de l'architecture couplé à l'instrumentation utilisée a été donné.

Chapitre IV : Mise en œuvre

IV-1 Introduction

Le but de ce chapitre est de modéliser le modèle dynamique et le modèle plateforme sur Simulink et implémenter sur STM32F303K8 pour le modèle dynamique et sur LPC1768 pour le modèle plateforme en utilisant Model Based design. Les deux implémentations seront interconnectées par la suite.

IV-2 Architecture générale

A l'origine, chaque modèle est implémenté sur un PC spécifique; le premier pour commander les vérins à travers le bus CAN et le second, connecté par CAN, était dédié au calcul des angles d'Euler ainsi qu'aux positions. Ces deux Pcs vont être remplacés par deux microcontrôleurs tel que décrit par l'architecture fonctionnelle donnée par la Figure IV.1

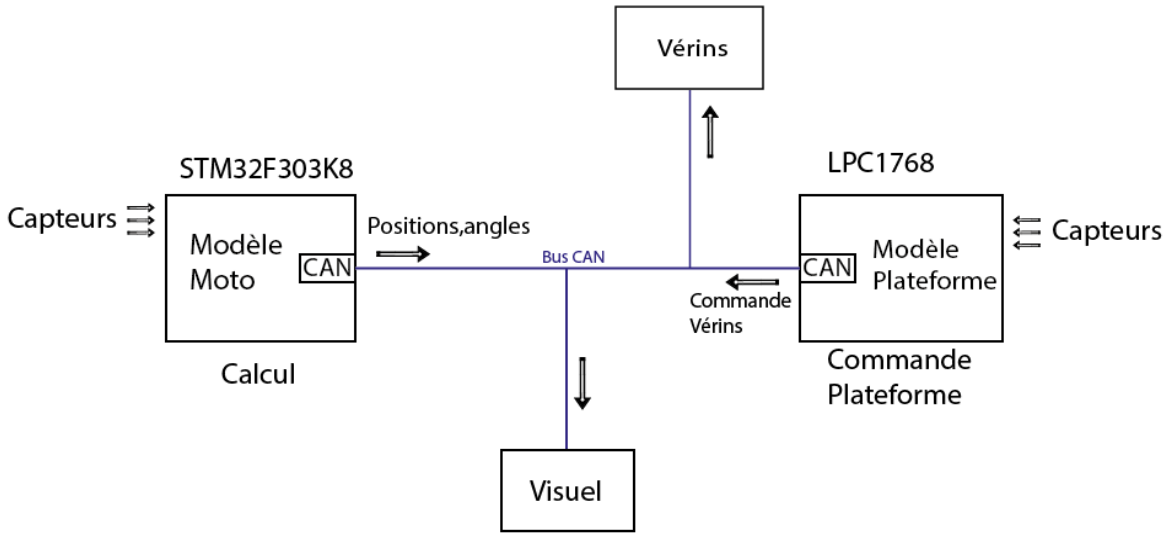

Figure IV.1 Architecture générale

IV-3 Modélisation et implémentation du modèle Dynamique

IV-3.1 Modélisation

La modélisation a été produite à travers l'équation (chapitre III) représentant le comportement dynamique de la moto aboutissant à la construction de la trajectoire de la moto. Ce modèle a été fait sous environnement Simulink.

Le bloc « Vélo Paramètres » est dédié aux paramètres du modèle « vélo ». Le bloc « Matrices » effectue le calcul des matricesM, C_1, K_0 et K_2 à partir des paramètres du modèle. Le bloc « Résolution » calcule les sorties du modèle (angle, vitesse et accélération de roulis et du guidon). Il calcule aussi la trajectoire de la moto, la boîte « Equilibre » calcule les couples en roulis et au guidon que doit faire le conducteur pour garder une inclinaison et un angle au guidon constants.

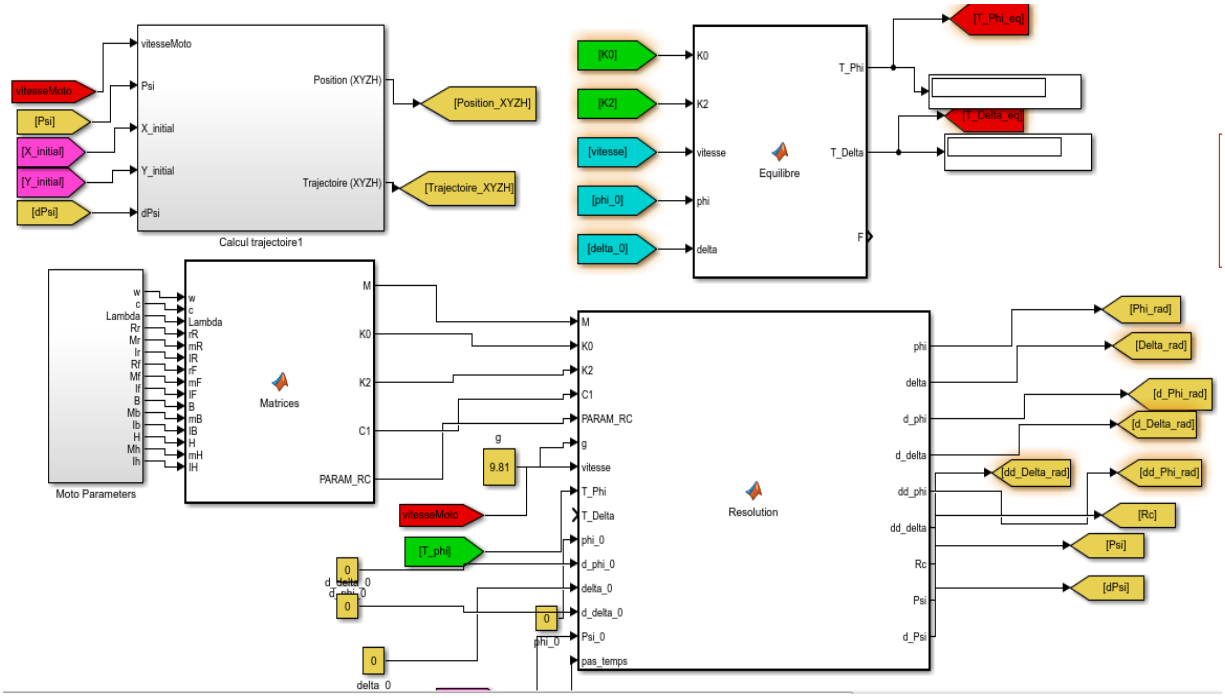

Figure IV.1 Modélisation du Modèle Dynamique

IV-3.2 Simulation

Le but de ce test est de vérifier si le modèle satisfait les principales exigences du système. Nous aborderons les trois formes de mise en œuvre à savoir MIL, SIL, PIL et HIL. Cette dernière forme constituera l'implémentation cible finale.

IV-3.2.1 MIL (Model in loop) :

On lance notre modèle par l'injection une vitesse de moto 90 Km/h et des angles guidon initiale défirent 0°,5°,10° et on vérifie la trajectoire obtenue (temps d'observation fixé à 10 sec).

Vitesse 90 Km/h angle guidon 0°			Vitesse 90 Km/h angle guidon 5°			Vitesse 90 Km/h angle guidon 10°		
Temps(s)	X(m)	Y(m)	Temps(s)	X(m)	Y(m)	Temps(s)	X(m)	Y(m)
0	0	0	0	0	0	0	0	0
1	25	0	1	24,92	1,96	1	24,68	3,93
2	50	0	2	49,61	5,86	2	48,44	11,71
3	75	0	3	73,92	11,69	3	70,69	23,12
4	100	0	4	97,71	19,39	4	90,87	37,87
5	125	0	5	120,82	28,92	5	108,50	55,60
6	150	0	6	143,13	40,21	6	123,16	75,85
7	175	0	7	164,50	53,18	7	134,52	98,12
8	200	0	8	184,82	67,74	8	142,32	121,87
9	225	0	9	203,97	83,81	9	146,41	146,53
10	250	0	10	221,86	101,28	10	146,71	171,53

Table IV.1 Valeurs trajectoire données par le modèle MIL pour différents angles

La représentation graphique de ces données (Figure IV.2) décrivent la trajectoire moto pour différents angles guidon

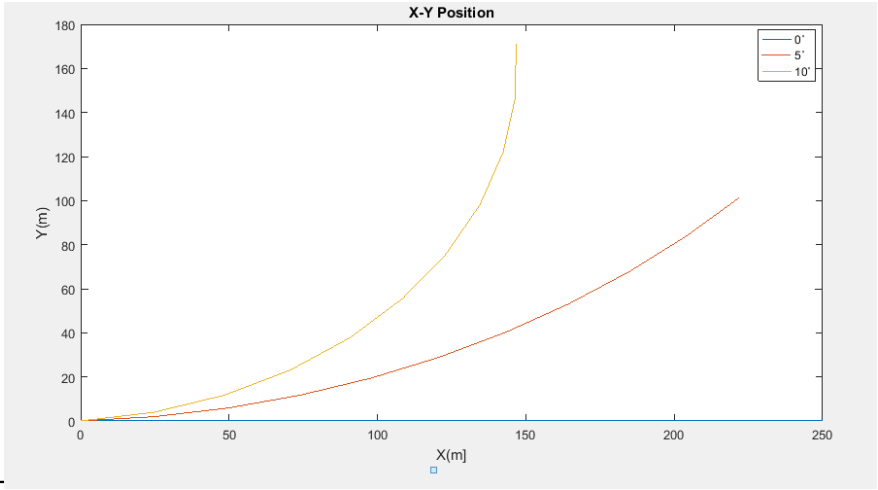

Figure IV.3 Trajectoire reconstruite via le modèle MIL

IV-3.2.2 SIL (Software in loop)

Avant qu'on de valider l'implémentation sur notre cible technologique, on doit vérifier que les Codes Généré par Embedded Coder représentant le modèle sont exécutable sans erreur. On peut tester ces codes sur le PC de développement.

Ce test consiste a générer des codes équivalents au modèle. Ce qui constitue la forme SIL. L'environnement est évidement virtuel.

Modèle SIL mis en œuvre est donné par la Figure IV.3

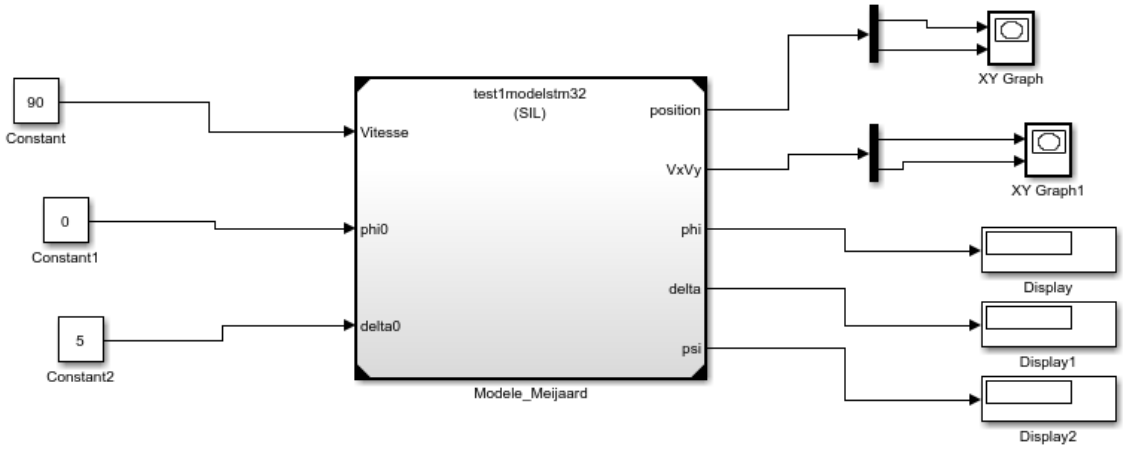

Figure IV.4 Modèle SIL

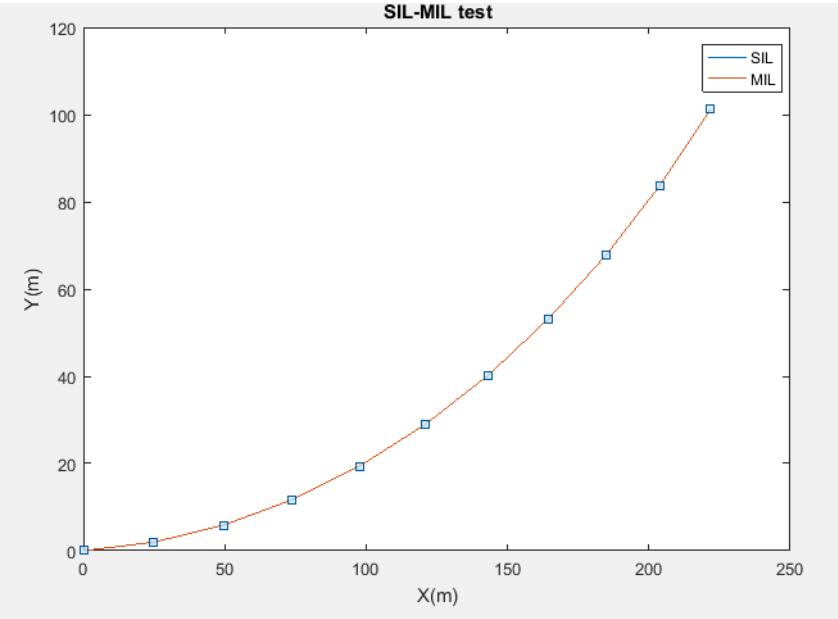

Figure IV.5 Trajectoires reconstruites comparées (MIL - SIL)

Rapport d'exécution des Codes

1. Summary

Total time (seconds × 1e-09)	70569
Measured time display options	('Units', 'Seconds', 'ScaleFactor', '1e-09', 'NumericFormat', '%0.0f')
Timer frequency (ticks per second)	2.501e+09
Profiling data created	06-Jun-2017 11:20:02

2. Profiled Sections of Code

Section	Maximum Execution Time	Average Execution Time	Maximum Self Time	Average Self Time	Calls	
test1modelstm32_initialize	6100	6100	6100	6100	1	
test1modelstm32_Init	55	55	55	55	1	
[+] test1modelstm32 [1 0]	12936	5856	2032	1431	11	

Le rapport d'exécution se divise en deux parties. La première partie donne les références temporelles pour un processeur tournant à une fréquence de 2.5GHz. Le code est exécuté en un temps de 70.569 µs. La deuxième partie explicite le modèle en fonctions équivalentes générées. On y trouve 3 fonctions (initialisation, init et la fonction principale). Les deux premières fonctions sont appelées une fois pour l'initialisation avec un Self Time correspondant au temps d'exécution sans utilisation de blocs externes au modèle (Graphique, scope, display...). Exécution time c'est le temps d'exécution incluant tous les blocs externes associé modèle. La fonction est appelée 11 fois alors le temps d'exécution d'un pas est 184.72 nS

IV-3.2.3 PIL (processor in the loop)

Ce test (Figure IV.4) consiste en la génération de codes qui son équivalent au modèle, sa compilation et son exécution pour notre le cible (STM32F303K8); le but étant de valider le code généré à travers sa comparaison avec les résultats du modèle MIL sans détection d'erreurs.

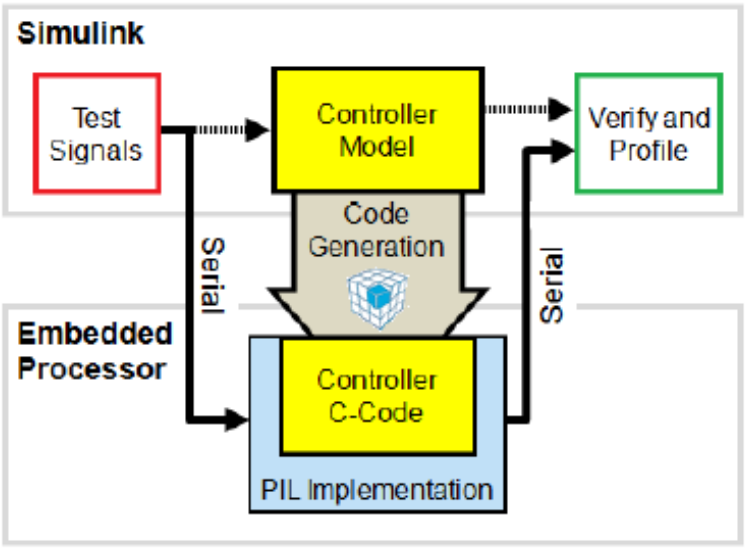

Figure IV.6 Modèle PIL mis en œuvre

L'environnement externe alimente le modèle PIL à travers la liaison série avec une vitesse de 115200 bauds. Les pins PA2 et PA15 pour UART y ont été réservées.

Les résultats obtenus sont données par les Figures IV.5 et IV.6

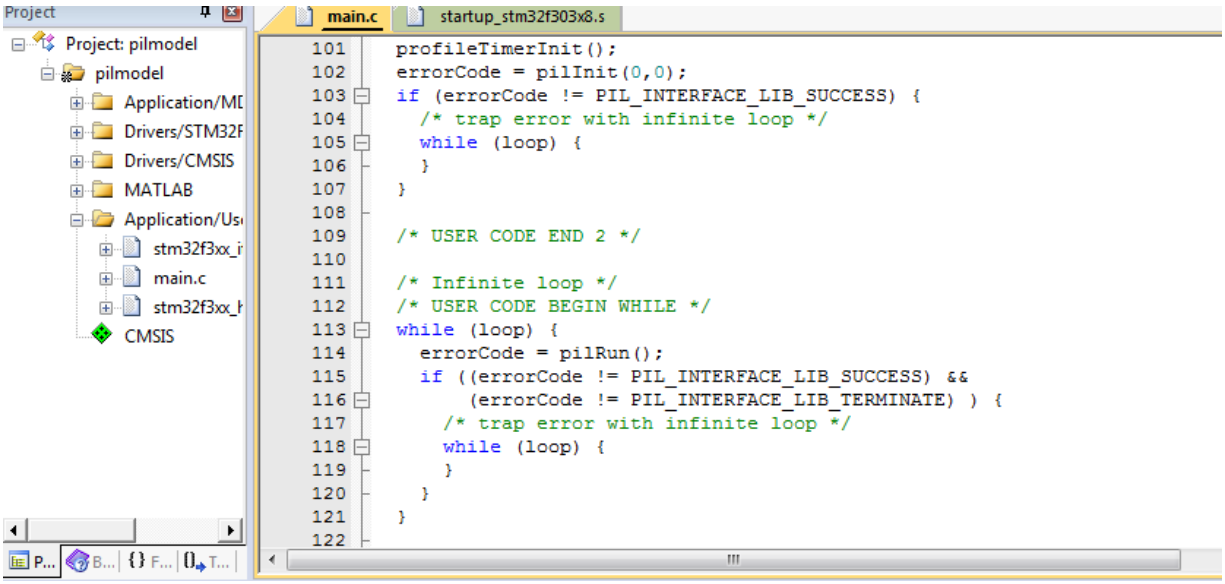

Figure IV. 5 Code généré par Embedded Coder

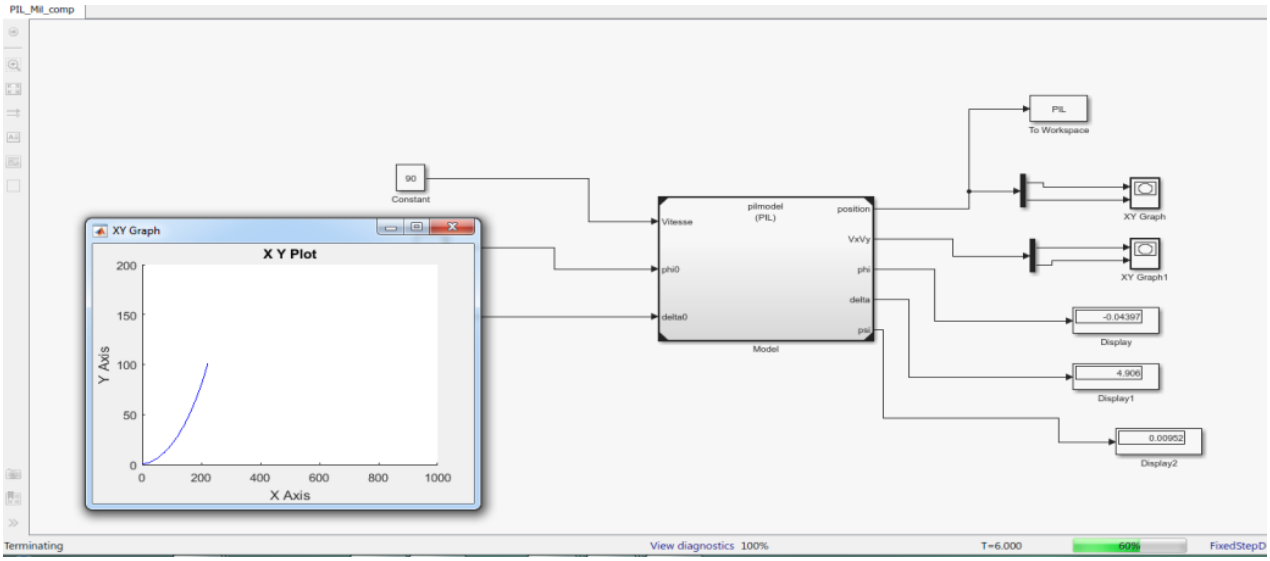

Figure IV. 6 Exécution sur la cible (angle de braquage de 5°)

Rapport d'exécution

1. Summary

Total time (seconds × 1e-09)	20243984
Measured time display options	('Units', 'Seconds', 'ScaleFactor', '1e-09', 'NumericFormat', '%0.0f')
Timer frequency (ticks per second)	6.4e+07
Profiling data created	06-Jun-2017 13:01:58

2. Profiled Sections of Code

Section	Maximum Execution Time	Average Execution Time	Maximum Self Time	Average Self Time	Calls
pilmodel_initialize	11125	11125	11125	11125	1
pilmodel_Init	1484	1484	1484	1484	1
pilmodel [1 0]	1893688	1839216	1893688	1839216	11

Ce code est exécuté sur le STM32F303K8 de fréquence 64 MHz en 20.243 ms. La fonction du modèle a été appelée onze fois. Le temps maximal d'une exécution est 1.893 ms. Et le temps d'exécution d'un pas est 172.15 µS ,On a vérifié que l'exécution du modèle sur la cible est sans erreur. La prochaine étape est de vérifier que les modes PIL et MIL sont équivalents. Les deux exécutions sont lancées en même temps pour une comparaison des deux modèles.

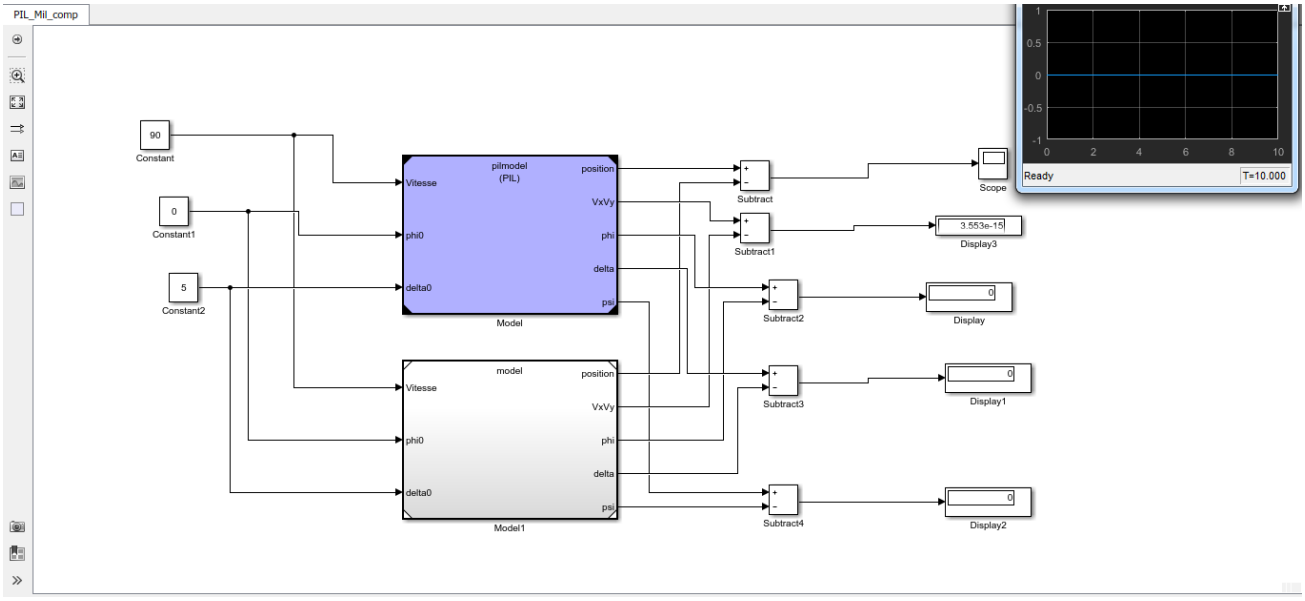

Figure IV.7 Exécution comparée

Comme le montre la Figure IV.7, la différence entre les deux modes d'exécution est nul. Le code généré par Embedded Coder est donc équivalent au modèle MIL.

IV-3.2.4 Mode HIL (Hardware in the loop)

Cette phase consiste à tester sur environnement réel. La transmission des données et la réception se fait via le module CAN (Controller area Network) avec une vitesse de 1 Mbit/sec exigée par le cahier de charge.
Sur Simulink, on utilise le package STM32-MAT/TARGET pour la génération des codes et STM32CubMX pour la configuration des périphérique et Keil 5 (ARM) pour la compilation et l'exécution sur la cible réelle.

Figure IV.8 Package STM32-MAT/TARGET sur Simulink

On utilise les blocs CAN (Send,Receive) pour la communication avec la cible pour la réception de la vitesse, de l'angle guidon et de la valeur du roulis initial. On récupère la trajectoire de la moto via une trame CAN. Elle a une taille de 8 octets. On récupère sur le CAN la position(x(m), y(m)) de la moto codifié chacun sur 4 octets.

Le tram de transmission : id0x100 : vitesse, phi, delta (angle roulis et angle guidon respectivement).

La trame de réception a un id de 0x118 portant les valeur des positions X, Y.

Remarque : Le bloc CAN_Send et CAN_Receive sur ce package ne fonctionnait pas correctement lorsque on envoie des trames qui contient des zéros sur le bloc. La suite de la trame est supprimée dès la réception d'un octet nul.

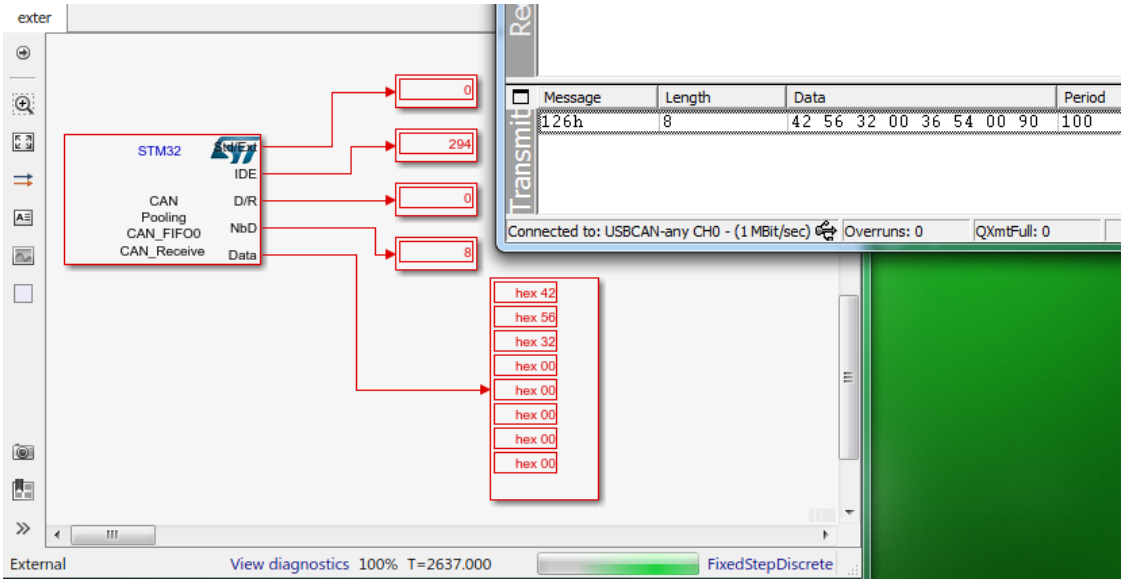

Figure IV.9 Problème de bloc CAN sur le Package STM32-MAT/TARGET

Solution

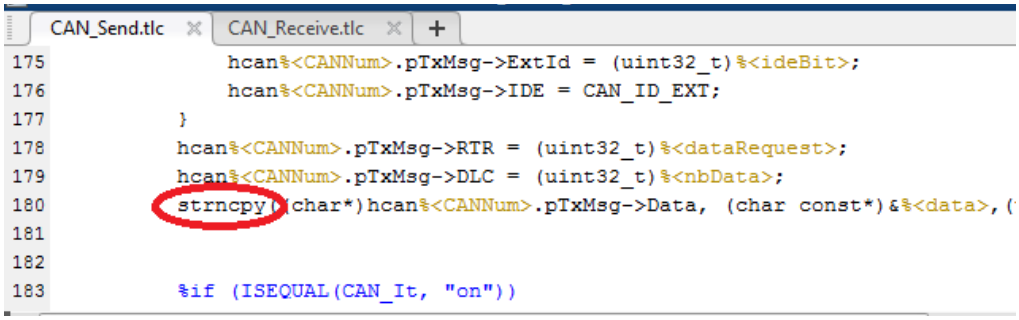

Figure IV.10 Source de bloc CAN_Send

```
CAN_Send.tlc   CAN_Receive.tlc   +
211     %<stand_ext> =(uint8_t)hcan%<CANNum>.pRxMsg->IDE;
212     %<dataRequest> =(uint8_t)hcan%<CANNum>.pRxMsg->RTR;
213     %<nbData> =(uint16_t)hcan%<CANNum>.pRxMsg->DLC;
214     //RP_MODIF %<data> =(uint8_t)hcan%<CANNum>.pRxMsg->Data;
215     strncpy((char*)&%<data>,(char const*)hcan%<CANNum>.pRxMsg->Data,
216
217     if(hcan%<CANNum>.pRxMsg->IDE == CAN_ID_STD){
218         %<ideBit> =(uint32_t)hcan%<CANNum>.pRxMsg->StdId;
```

Figure IV.11 Source de bloc CAN_Receive

Ce problème est due au l'utilisation des fonctions de traitement de chaines de caractères.

```
void mystrncpy(char * reciver , char const* source,int len)
{
  for(int i =0;i<len;i++)
   *(reciver +i) =*(source+i);
}
```

Figure IV.12 Utilisation des fonctions de traitement de chaines de caractères

Résultat

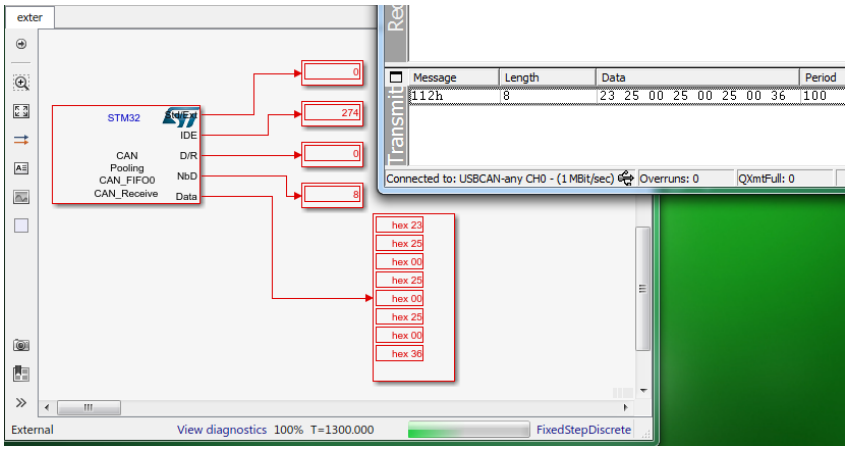

Figure IV.12 le bloc CAN après correction

On lance le modèle en mode externe et on envoie par CAN la vitesse et l'angle guidon et roulis initiale et on récupère la position par via le bus CAN.

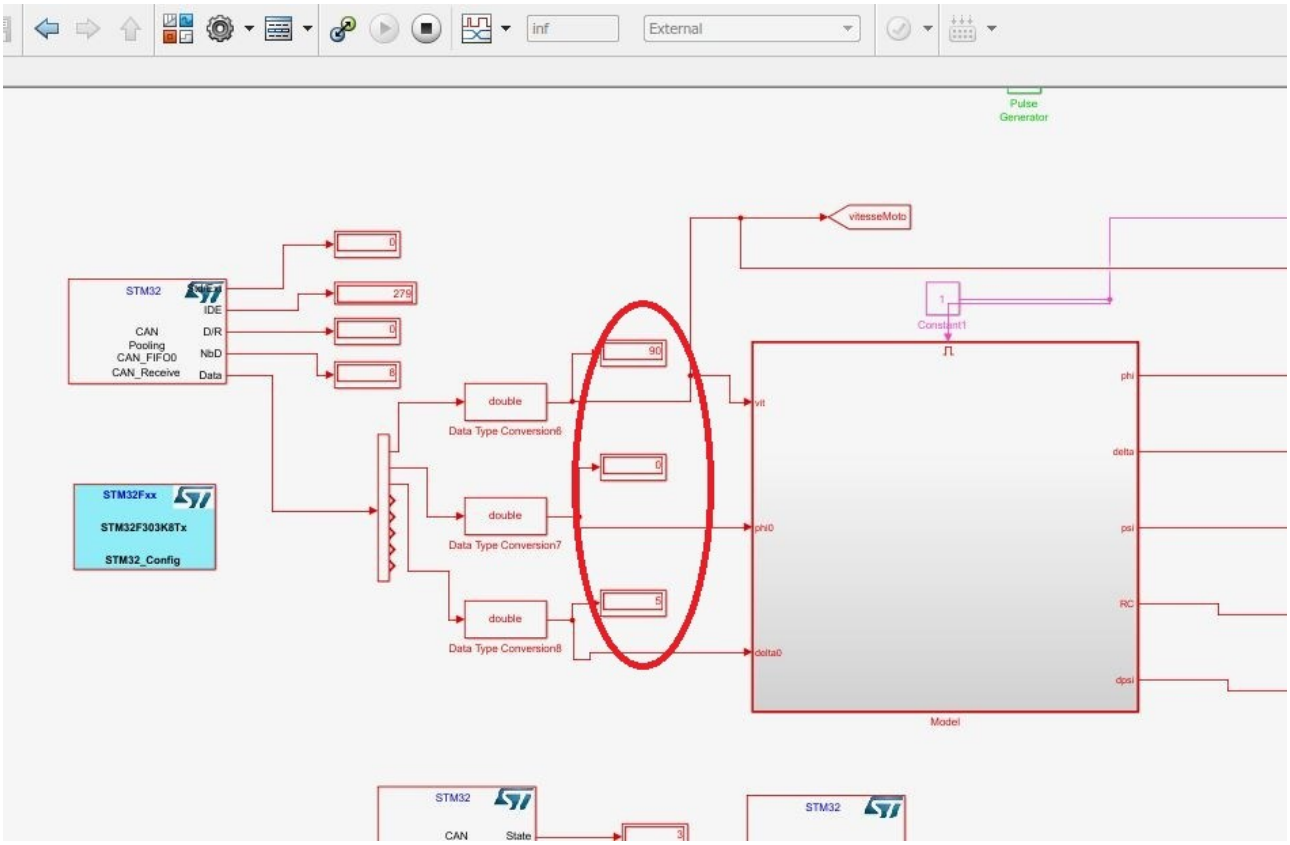

Figure IV.13 Réception de vitesse et les angles initiaux

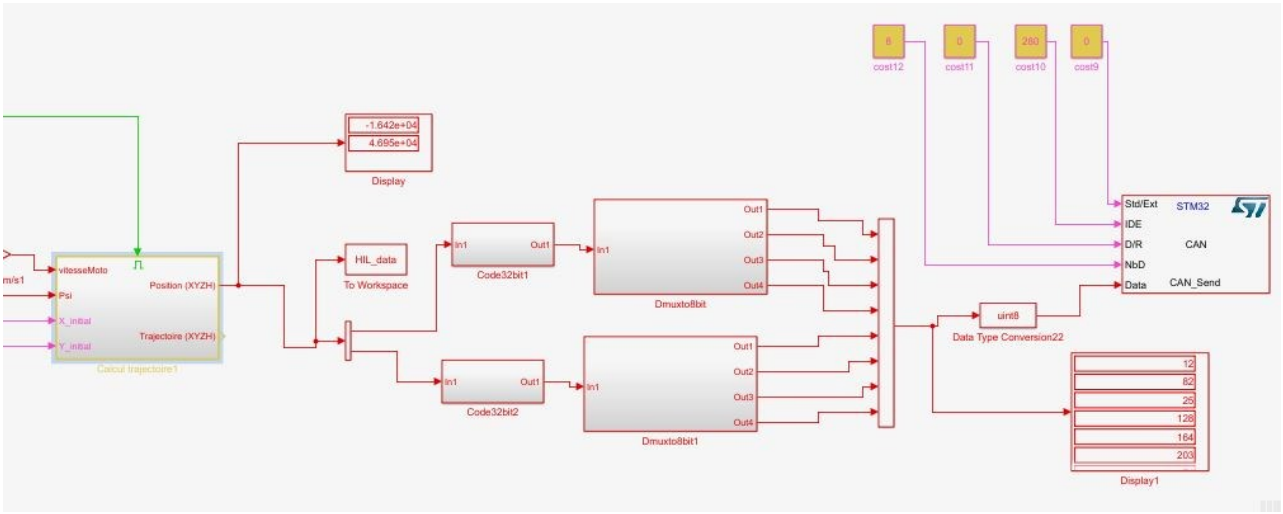

Figure IV.14 Transmission de la position

Avant la transmission de la position, on a codifié chaque position sur 4 octets (32 bits)

Un bit pour le signe et 31 bits pour la donnée. On divise la trame en 8 bits pour construire la trame CAN. La figure IV.15 montre le trafic du bus CAN correspondant.

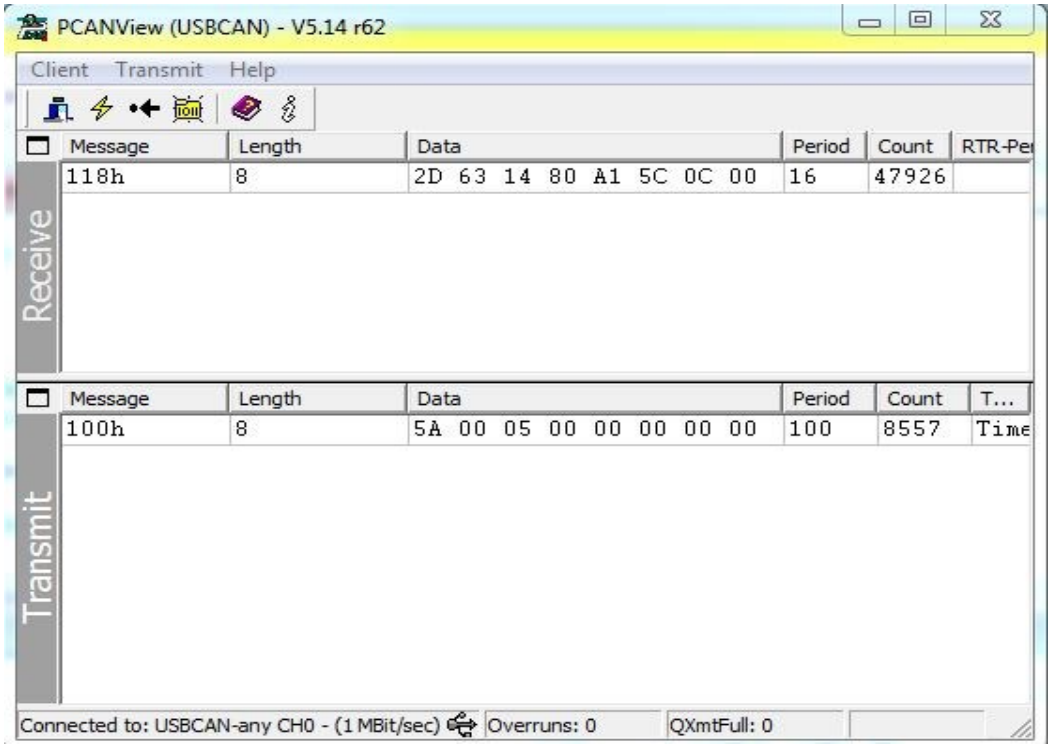

Figure IV.15 Le trafic dans le bus CAN

Remarque : pour que le bloc CAN recevez fonctionne i faut ajouter un filtre de trafic sur le code généré par Embedded Coder « le code sur l'annexe »

IV-3.3 Evaluation des performances

Le rapport de génération de code contient la liste des fichiers générés. Le ficher de type (nom model .c) contient la fonction principale du modèle qui est appelée nom modèle _step(). Une traçabilité entre le ficher et le modèle est possible permettant la remontée vers le bloc Simulink mis en jeu.

On mesure le temps d'exécution d'un pas du modèle dynamique seul sur la cible et on doit s'assurer que le modèle fonctionne à 100 Hz (exigé par cahier de charge). On utilise pour cela un Timer pour mesurer le temps d'exécution d'un pas.

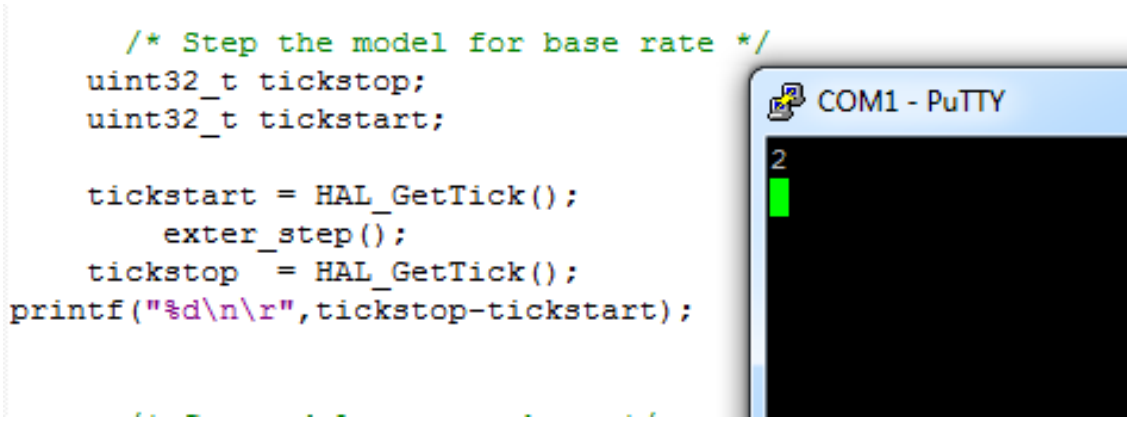

Figure IV.16 Mesure le temps d'exécution d'un pas

On trouve 2 ms pour l'exécution d'un pas du modèle sans la transmission et la réception des données. Lorsque l'on ajoute les blocs CAN transmission et réception, on trouve un temps de 46 ms.

IV-4 Modélisation et implémentation du modèle de la Plateforme

Le modèle Plateforme reçoit les données générées par le modèle dynamique à savoir la position XYZ et les angles roulis tangage et lacet. Il reçoit en plus les données des capteurs couple guidon, du démarreur, contact, klaxon, comodo, ... Il reçoit aussi les positions actuelles des vérins.

A partir de c'est valeur, le modèle calcule les élongations des vérins ainsi que la vitesse de déplacement pour la restitution de mouvement de la moto. La transmission et la réception des données ce fait par CAN et la cible utilisées pour l'implémentation du modèle est le circuit ARM Cortex M0 LPC1768.

Simulink ne disposant pas des blocs périphérique de cette cible, nous étions emmené à les construire en utilisant l'approche S-FunctionBuider . Les blocs implémentés sont le CAN Send, le CAN Receive, Printf to série.

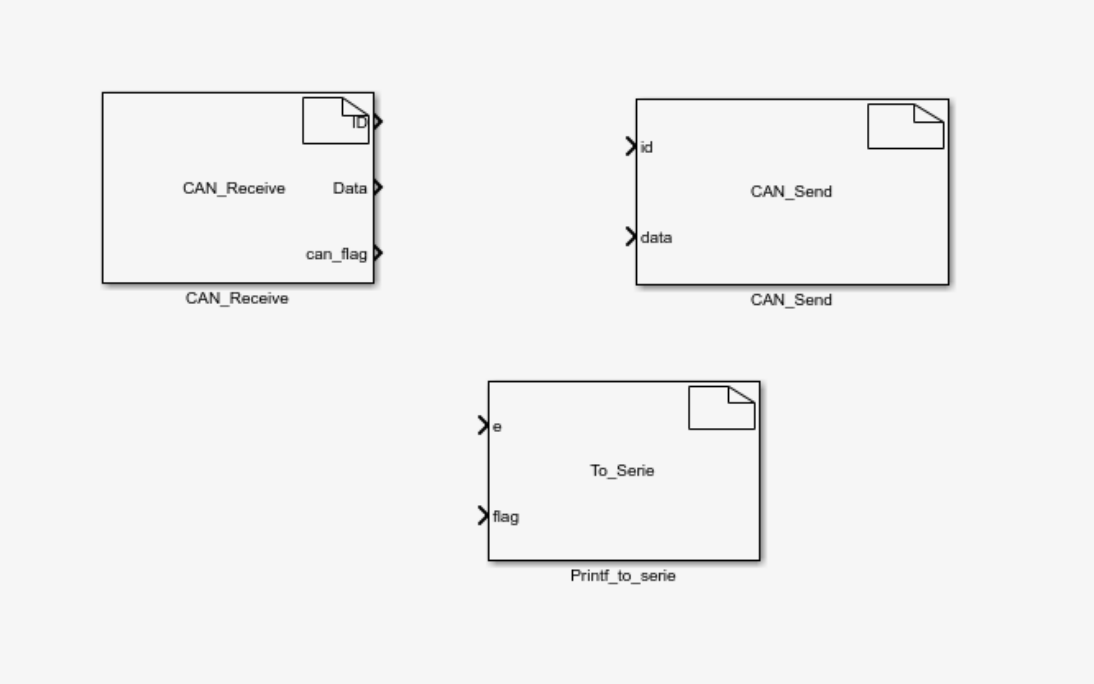

Figure IV.17 Développement des blocs inexistants

Pour cette partie, on ne peut pas faire tous les tests (PIL, SIL) car le package « ARM Cortex-M Support from Embedded Coder » ne permet pas la génération de codes pour tester sur le cible. On ne pourra donc faire qu'une comparaison entre l'exécution du modèle (MIL) et l'exécution sur la cible (HIL).

IV-4 -1 Modélisation

Le modèle plateforme ce compose en trois parties à savoir le bloc principale 'Restitution vérins qui a des entrées arrivant du modèle dynamique (position et angle roulis) par CAN et des entres des capteurs (position actuelle des vérins, accélérateur, frein).

Le deuxième bloc 'Réception informations' reçoit les trames CAN qui vont être filtrées puis séparées pour ne tenir compte que d'identificateurs décrits sur le tableau suivant :

Les trames CAN utilisé

	ID Trame	Octet(s)	Capteur
Volant	231H	1&2	8
Accélérateur		3	16
Frein AR		4	17
Embrayage		5	18
Frein AV		6	19
Courant		7	20
Commodo 1	232H		48

Commodo 2			50
Effort D	234H		22
Effort G			23
Courant PF	235H		176
Courant pf			177

Table IV.1 Les trames CAN

Commodos et trame 232 H		
Octet	Bit	Description
1	0 à 2	Rapport de BV (valeurs de 0 à 7)
	6	Démarreur
	7	Neiman
2	0	Boutonadditionnel orange
	1	Phare
	2	Lave-glace / Bouton additionnel violet
	3	Klaxon
	4	Cligno D
	5	Cligno G
	6	Codes
Efforts		
Octet	Bit	Description
1	/	Capteur D
2	/	Capteur G
Courant		
1..2	/	PF
3..4	/	Pf

Table IV.2 Les trames CAN

Le troisième bloc (Réception CAN) reçoit les trames CAN à partir de modèle dynamique et il va séparer les trames pour avoir le positon, la vitesse et les angles

Trame : 0x117 pour la sortie info moteur

Trame : 0x118 pour la sortie XYZ

Trame : 0x119 pour la sortie roulis

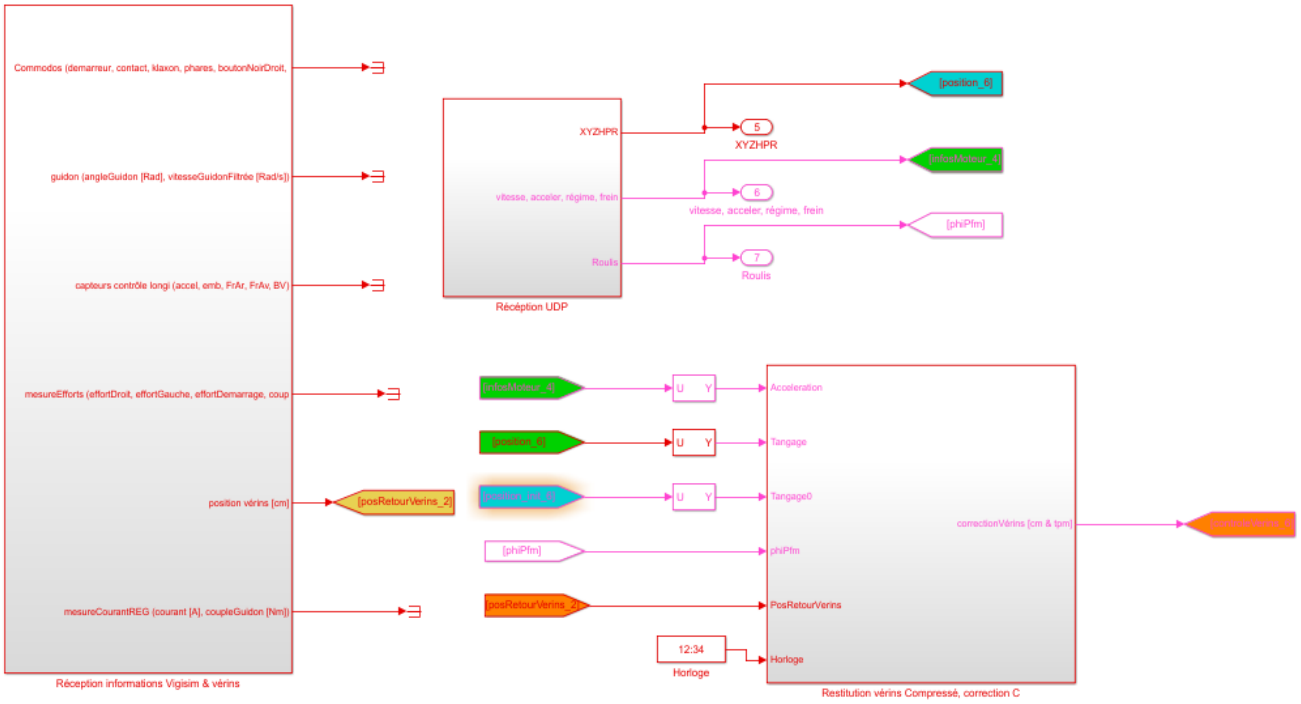

Figure IV.17 Modèle de la plateforme sur Simulink

IV-4 -2 Simulation et implémentation

Pour vérifier le fonctionnement du modèle, on a comparé les réponses indicielles entre le modèle MIL et le HIL.

Message	Length	Data	Period	Count	Trigger
117h	8	00 00 00 00 17 70 00 00	100	11170	Time
118h	8	00 64 00 00 00 64 00 00	100	19028	Time
119h	8	00 01 00 01 00 01 00 01	100	11005	Time

COM13 - PuTTY

```
message :  -6.000000,  6.000000,  -0.043780,  0.000000,  3519.529118,  3519.529118,  0.000000,  2000.000000,
message :  -6.000000,  6.000000,  -0.042687,  0.000000,  3549.857614,  3549.857614,  0.000000,  2000.000000,
message :  -6.000000,  6.000000,  -0.041651,  0.000000,  3579.247972,  3579.247972,  0.000000,  2000.000000,
message :  -6.000000,  6.000000,  -0.040669,  0.000000,  3607.729211,  3607.729211,  0.000000,  2000.000000,
```

Figure IV.18 Envoi des données vers CAN et récupération sur le port série

Résultat

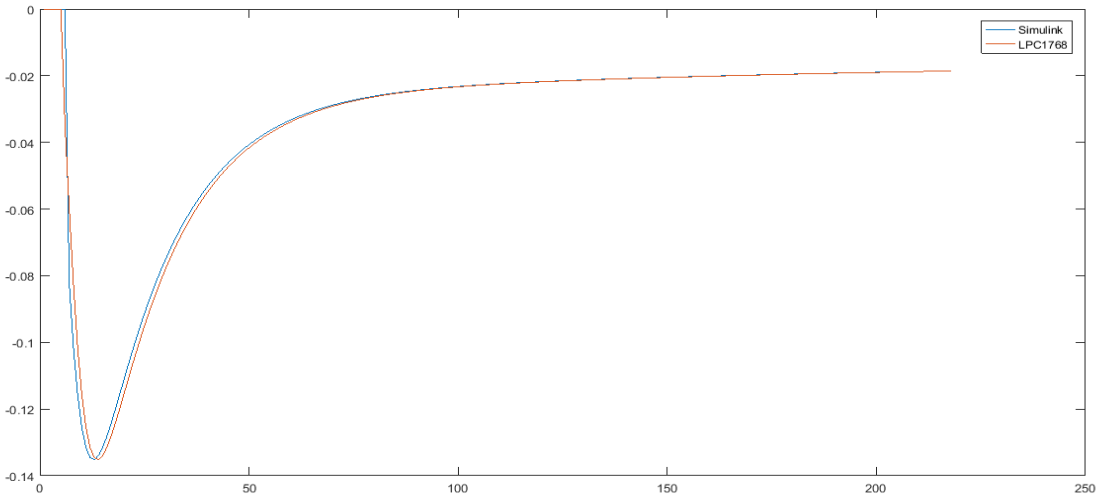

Figure IV.19 Position vérin fonctionnements sur Simulink et sur LPC1768

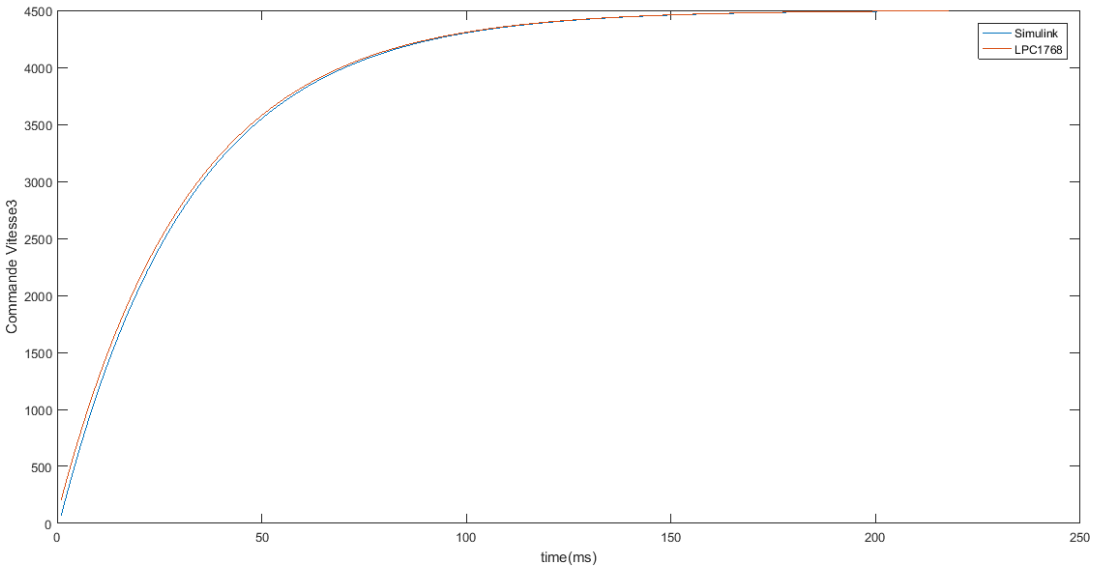

Figure IV.20 Commande vitesse MIL / HIL (LPC1768)

Un retard très faible est constaté entre les deux sorties; les amplitudes restant identiques.

IV-4 -3 Evaluation des performances

Dans le cas du HIL, on mesure le temps d'exécution d'un pas par l'utilisation du Timer de LPC1768. Le résultat est donnée par la Figure IV.21.

```
Timer t;
 t.start();
 simuMoto_Aout_2014_3_step();
 t.stop();
printf("The time taken was %d ms\n\r", t.read_ms());
```

```
The time taken was 1 ms
```

Figure IV.21 Mesure de temps d'exécution

Ce temps est estimé à 1 ms, le modèle pouvant donc s'exécuté à une fréquence de 1kHz.

IV-5 Fonctionnement HIL combiné Modèle dynamique / Modèle de la plateforme

L'étape finale est de rassembler les deux systèmes (Figure IV.22) et voir la communication sur le bus CAN (Figure IV.23) telle que les sorties du modèle de la plateforme (position et vitesse). Ces dernières sont codées sur 32 bits. On a besoin donc de 4 identificateurs supplémentaires (id0x40 ...id0x43) pour transmettre ces données. On visualise sur le bus les trames envoyées par le modèle dynamique avec les deux identificateurs (id0x117 et id0x118). On envoie via la trame d'identificateur id0x119 les données informations moteur (Figure IV.23).

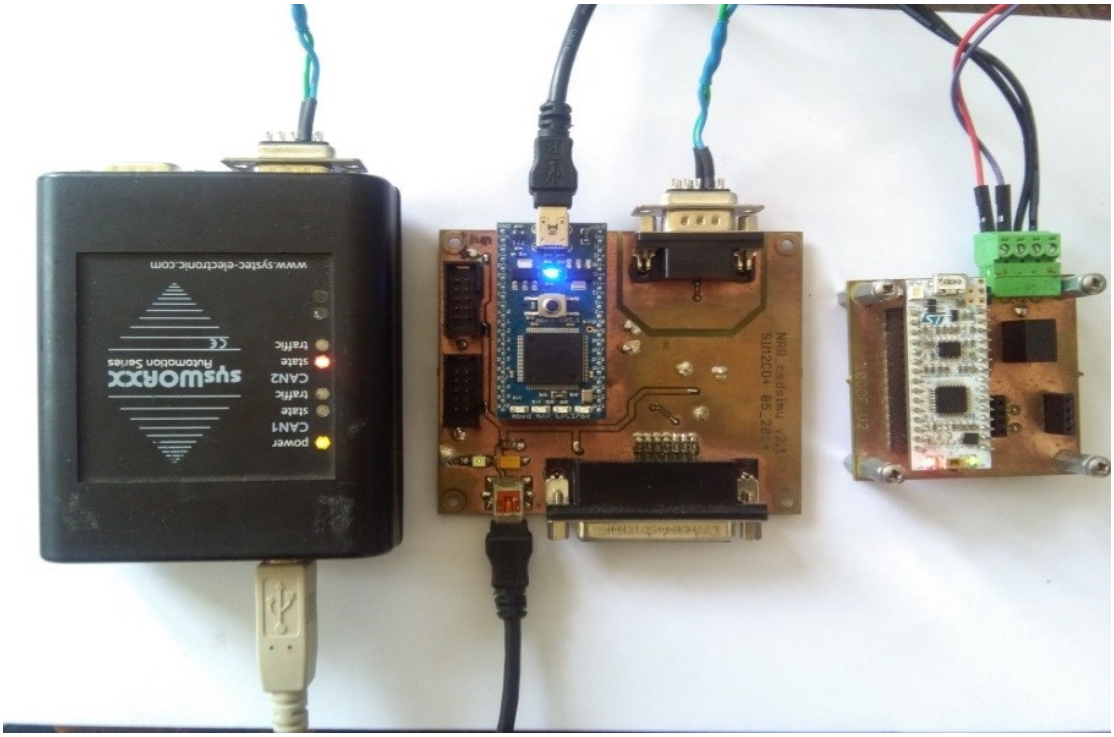

Figure IV.22 modèle dynamique + modèle de la plateforme

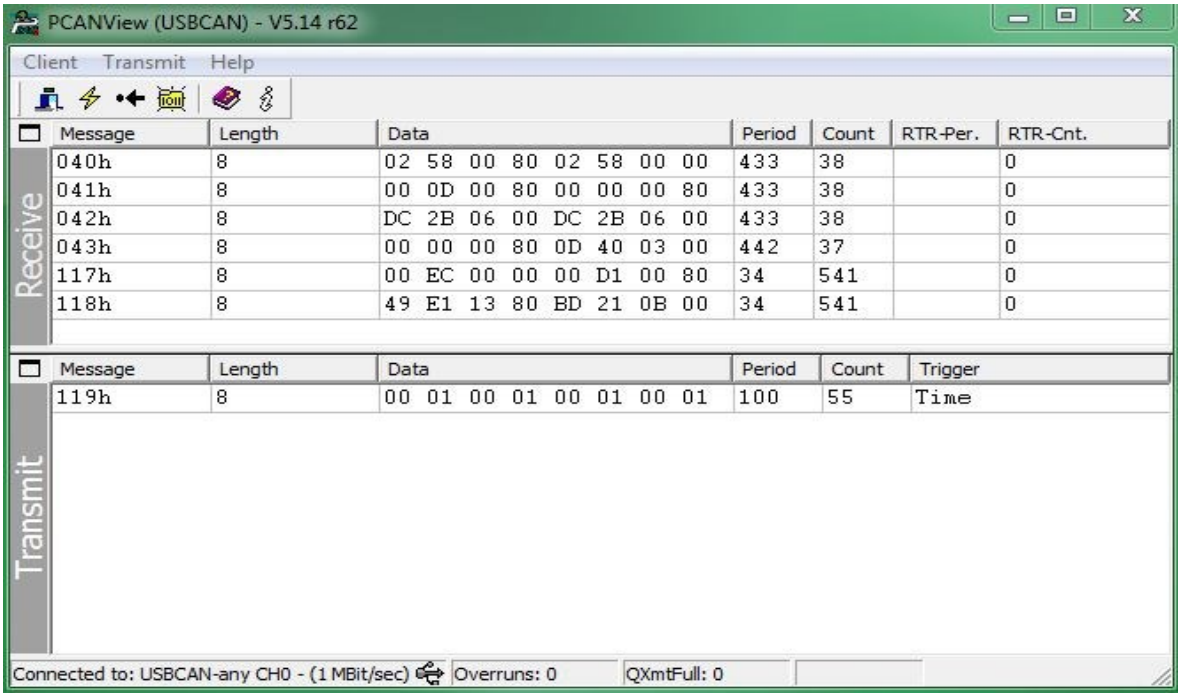

Figure IV.23 les trames sur le but CAN pour le modèle final

IV-6 Optimisation

Il existe un outil sur Simulink « Model Advisor » qui nous permet d'optimiser le code généré par Embedded Coder par rapport à plusieurs critères. On optimise notre code par rapport l'occupation mémoire comme gain supplémentaire.

On considère les paramètres suivants :

Total RO Size : C'est la taille du code seulement
Total RW Size : La Taille minimale de RAM nécessaire pour exécuter le code.
Total ROM Size : C'est la taille du code et des données qui doivent être stockées en Mémoire flash. On a besoin au moins de cette mémoire flash Disponible.

IV-6.1 Modèle dynamique

Sans optimisation	Après optimization
RO Size = 19368 (18.91 KO) RW Size = 2080 (2.031 KO) ROM Size =19770 (18.98 KO)	RO Size = 16.14 KO RW Size = 1.92 KO ROM Size = 16.17 KO

Table IV.3 comparaison entre l'occupation mémoire pour le Modèle dynamique

IV-6.1 Modèle de la plateforme

Avant optimization	Sans optimization
RO Size = 61.03 KO RW Size = 5.92 KO ROM Size =62.59 KO	RO Size: 38.51 KO RW Size: 2.06 KO ROM Size: 38.61 KO

Table IV.3 comparaison entre l'occupation mémoire - Modèle de la plateforme

IV-7 Conclusion

On a implémenté les deux modèles et on a vérifié l'équivalence entre le fonctionnement sur Simulink (mode MIL) et sur la cible (HIL). La communication entre calculateurs est fonctionnelle via le bus CAN. Pour avoir une synchronisation entre les deux modèles, on a ajouté un retard de 2 ms pour le modèle de la plateforme pour qu'elle fonctionne correctement avec le modèle dynamique.

Le système fonctionne à 333 Hz, 3 fois plus grande que l'exigence du cahier de charge.

Conclusion générale

Grâce à ce travail, nous avons appris et acquis une expérience dans deux domaines d'application principaux: Model-Based Design pour les cibles embarqués et les Simulateurs à deux roues motorisées.

Nous avons découvert dans Simulink un ensemble d'outils puissants qui offre la simulation au niveau système, génération automatique de code, ainsi que le test et la vérification.

Nous avons présenté des différents Méthodologies de conception et les cycles de développement, on a choisi la Méthodologies Model-Based Design qui est baser sur le cycle en V, après on a présenté plusieurs simulateurs a deux roues motorisées pour situer le simulateur d'IFSTAR comme objet de notre étude.

On a appliqué le processus Model-Based Design pour implémenter le modèle dynamique sur le microcontrôleur STM32F303K8 et implémenté le modèle de la plateforme sur le microcontrôleur ARM LPC1768. On a couplé les deux systèmes et vérifier la condition de cahier de charge que le système fonctionne à 100 Hz.

Bibliographie

[1] NEHAOUA Lamri. Conception et réalisation d'une plateforme mécatronique dédiée à la simulation de conduite des véhicules deux-roues motorisés. Université d'Evry-Val d'Essonne Laboratoire d'Informatique, Biologie Intégrative et Systèmes Complexes ; 10 Décembre 2008.

[2] J. P. Meijaard, Jim M. Papadopoulos , Andy Ruina, A. L. Schwab. Linearized dynamics equations for the balance and steer of a bicycle : a benchmark and review. School of MMME, The University of Nottingham, University Park, Nottingham NG7 2RD, UK ; 9 June 2007.

[3] Damien Foures. Validation de modèles de simulation. l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier), 26 Juin 2015.

[4] Simon Abourida, Christian Dufour, Jean Bélanger. Real-Time and Hardware-In-The-Loop Simulation of Electric Drives and Power Electronics : Process, problems and solutions. The 2005 International Power ElectronicsConference.

[5] L. Nehaoua, S. Hima, H. Arioui, N. Séguy and S. Éspié. Design and Modeling of a New Motorcycle Riding Simulator. Proceedings of the 2007 American Control Conference, New York City, USA, July 11-13, 2007.

[6] Salim Maakaroun Modélisation et simulation dynamique d'un véhicule urbain innovant en utilisant le formalisme de la robotique. Automatique / Robotique. Ecole des Mines de Nantes, 2011.

[7] RM0316 Reference manual : STM32F303xB/C/D/E, STM32F303x6/8, STM32F328x8, STM32F358xC, STM32F398xE advanced ARM ʀ -based MCUs, http://www.st.com

[8] UM 1786 User Manual : Description of STM32F3 HAL and Low Layer drivers, http://www.st.com

[9] UM 1718 User manual : STM32CubeMX for STM32 configuration and initialization C code generation, http://www.st.com

[10] STM32F302x6 STM32F302x8 : ARM ʀ Cortex ʀ -M4 32-bit MCU+FPU, up to 64 KB Flash, 16 KB SRAM, ADC, DAC, USB, CAN, COMP, Op-Amp, 2.0 - 3.6 V,

http://www.st.com

[11] Webinars : MATLAB -MathWorks,
https://www.mathworks.com/products/matlab/webinas.html

[12] Model-Based Design :
https://www.mathworks.com/solutions/model-based-design.html

[13] CAN Bit Time Calculation,

http://www.bittiming.can-wiki.info

Annexe

Filtre CAN

```
CAN_FilterConfTypeDef sFilterConfig;
 sFilterConfig.FilterNumber = 0;
 sFilterConfig.FilterMode = CAN_FILTERMODE_IDMASK;
 sFilterConfig.FilterScale = CAN_FILTERSCALE_32BIT;
 sFilterConfig.FilterIdHigh = 0x0000;
 sFilterConfig.FilterIdLow = 0x0000;
 sFilterConfig.FilterMaskIdHigh = 0x0000;
 sFilterConfig.FilterMaskIdLow = 0x0000;
 sFilterConfig.FilterFIFOAssignment = 0;
 sFilterConfig.FilterActivation = ENABLE;
 sFilterConfig.BankNumber = 14;

if (HAL_CAN_ConfigFilter(&hcan, &sFilterConfig) != HAL_OK)
 {
 /* Filter configuration Error */
 Error_Handler();
 }
```