REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
الجمهورية الجزائرية الديمقراطية الشعبية

MINISTRY OF HIGHER EDUCATION
AND SCIENTIFIC RESEARCH

HIGHER SCHOOL IN APPLIED SCIENCES
–T L E M C E N–

ESSA Tlemcen-تلمسان

المدرسة العليا في العلوم التطبيقية
École Supérieure en Sciences Appliquées

وزارة التعليم العالي والبحث العلمي

المدرسة العليا في العلوم التطبيقية
- تلمسان -

Mémoire de fin d'étude

Pour l'obtention du diplôme d'Ingénieur

Filière : Electronique
Spécialité : Informatique Industrielle

Présenté par : Ahmed Khodhir REZIG et Sid Ahmed LAKEHAL

Thème

Développement d'un dispositif embarqué de collecte des données CAN Bus pour le diagnostic automobile via le protocole OBD-II

Soutenu publiquement,le 20 / 06 / 2026 , devant le jury composé de :

Dr BELLOUFA Fayssal	MCB ESSAT	Président
Dr MESSIRDI Mohammed	MCB Université Mascara	Directeur de mémoire
Dr OUDJAMA Farid	MCB ESSAT	Co- Directeur de mémoire
Dr KARAOUZENE Zoheir	MCB ESSAT	Examinateur 1
Dr MERAD Fayza	MRB CDS Oran	Examinateur 2
M ABDELLAOUI Abdou	Conseillé Diagnostic Automobil	Invité
Dr OUHOUD Amina	MCA ESSAT	CDE
Dr MIMOUNI Faiza	MCA ESSAT	CATI

Année universitaire : 2025 /2026.


بسم الله الرحمن الرحيم

Dédicace

Je dédie d'abord ce travail à moi-même, pour la patience, la persévérance et la détermination qui m'ont permis de surmonter les difficultés de ce parcours. Pour chaque défi affronté, chaque moment de doute dépassé et chaque effort qui m'a conduit jusqu'ici, je suis fier du chemin accompli.

À ma chère mère, dont l'amour inconditionnel, les prières, les sacrifices et l'attention constante ont toujours été ma plus grande source de force. Aucun mot ne saurait exprimer pleinement ma gratitude pour tout ce que tu as fait pour moi et pour le soutien que tu m'as apporté tout au long de ma vie.

À mon père, **Ammar**, qui a toujours répondu à mes besoins, m'a soutenu et a veillé à ce que je ne manque de rien. Ta générosité, ta force et ta présence constante m'ont donné la confiance et la stabilité nécessaires pour avancer. Cette réussite est aussi la tienne.

À mon frère, **Aness**, et à mes sœurs, **Rihab, Manar et Loudjaine**, pour leur affection et leurs encouragements, ainsi qu'à la petite **Amira**, qui apporte de la joie à notre famille.

À mes grand-mères, dont les prières et l'amour m'ont accompagné tout au long de ma vie, et à la mémoire de mes deux grands-pères, qu'Allah leur fasse miséricorde. Leurs valeurs et leur héritage resteront toujours vivants au sein de notre famille.

À tous mes oncles et toutes mes tantes, pour leur affection, leurs conseils et leur soutien, et tout particulièrement à mon oncle, **Mohammed Lamine**, qui a toujours été pour moi bien plus qu'un oncle : un grand frère, un guide et une personne sur laquelle je peux compter.

À tous les membres de ma famille, où qu'ils se trouvent, pour leur amour et leurs encouragements constants.

À tous mes camarades de promotion, avec qui j'ai partagé cinq années inoubliables d'efforts, de défis, de rires, de réussites et de souvenirs. Plus particulièrement à **Sid Ahmed**, mon binôme, mon ami et mon frère tout au long de ce parcours, ainsi qu'à **Aboubaker, Yacine, Youcef, Amine, Aissam, Habib, Salah, Mohammed et Abderahmane**, dont l'amitié a rendu cette aventure plus précieuse et inoubliable.

À mes amis de la résidence universitaire, pour les moments de partage et les souvenirs vécus durant ces années.

À mes amis d'enfance, **Yacine HADDAD, Abdelmoez MOULAY, Wassim ZOUAI et Mohammed El Habib REZGANE**, dont l'amitié sincère a traversé les années.

Enfin, je dédie ce travail à toutes les personnes qui ont cru en moi, m'ont soutenu et ont contribué, de près ou de loin, à la personne que je suis devenu.

J'ai consacré à ce projet le meilleur de mes efforts, de mon savoir et de ma détermination. Puisse-t-il demeurer comme la trace de mon passage, le témoignage qu'un jour, Ahmed était ici, qu'il a imaginé, construit et laissé derrière lui quelque chose qui lui survivra.

Ahmed Khodhir REZIG

Dédicace

Je dédie d'abord ce travail à moi-même, pour la patience, la persévérance et la détermination qui m'ont permis de surmonter les difficultés de ce parcours. Pour chaque défi affronté, chaque moment de doute dépassé et chaque effort qui m'a conduit jusqu'ici, je suis fier du chemin accompli.

À ma chère mère, **K. BENZINA**, mon refuge et ma plus belle force. Ton amour inconditionnel, tes prières silencieuses, tes sacrifices et ta tendresse ont été ma lumière dans les moments les plus sombres. Je ne trouverai jamais les mots justes pour exprimer tout ce que je te dois. Si j'ai pu arriver jusqu'ici, c'est avant tout grâce à toi.

À mon père, **Ahmed LAKEHAL**, dont la présence discrète mais constante m'a toujours guidé. Ta force, ton courage et ton engagement pour notre famille m'ont appris la valeur du travail et de la dignité. Cette réussite est autant la tienne que la mienne.

À mon frère, **Ilyes**, et à mes sœurs, **Amel et Kaouther**, pour leur amour sincère, leur soutien et leur présence qui ont rendu chaque étape plus légère.

À tous les membres de ma famille, pour leur affection, leurs prières et leur soutien, même à distance.

À mes camarades de promotion, avec qui j'ai partagé des années riches en émotions, en efforts et en souvenirs inoubliables. Ces années resteront gravées en moi. Une pensée particulière à **Ahmed Khodhir**, mon binôme, mon ami et mon frère de parcours, ainsi qu'à **Aboubaker, Yacine, Youcef, Amine, Aissam, Habib, Salah, Mohammed et Abderahmane**, qui ont rendu cette aventure plus humaine et plus belle.

Enfin, je dédie ce travail à toutes les personnes qui ont cru en moi, parfois plus que je ne croyais en moi-même, et qui m'ont soutenu sans attendre en retour. Leur présence, même silencieuse, a compté plus que tout.

Ce travail est bien plus qu'un simple mémoire. C'est le fruit de nuits de doute, de jours d'effort, de sacrifices invisibles et de rêves tenaces. Qu'il reste comme une trace de mon passage, la preuve qu'un jour, Sid Ahmed a essayé, a persisté, et a construit quelque chose de lui-même.

Sid Ahmed LAKEHAL

Remerciements

Au nom de Dieu, le Tout-Puissant, le Clément, le Miséricordieux, qui nous a encouragés à rechercher le savoir et la science, et qui nous a dotés des facultés et des moyens nécessaires pour y parvenir. Nous Lui rendons grâce pour la volonté, le courage, la patience et la force qu'Il nous a accordés tout au long de ces années d'études.

Nous exprimons notre profonde gratitude et nos sincères remerciements à notre encadrant, Monsieur **Mohammed MESSIRDI**, pour sa confiance, son soutien constant et sa grande patience tout au long de ce projet. Vous avez su nous guider avec bienveillance et générosité, et avez été bien plus qu'un enseignant : un véritable mentor. Grâce à votre expérience dans la recherche et l'enseignement, ainsi qu'à vos précieux conseils, nous avons pu découvrir et approfondir le monde de la recherche scientifique dans le domaine du diagnostic automobile.

Nous remercions également chaleureusement Monsieur **Farid OUDJAMA**, notre co-encadrant, pour sa disponibilité, sa patience et ses précieux conseils tout au long de ce travail.

Nous tenons à remercier l'ensemble des enseignants du département d'Électronique de l'École Supérieure en Sciences Appliquées de Tlemcen pour les connaissances et les compétences qu'ils nous ont transmises au fil des années.

Nos pensées reconnaissantes vont également à toutes les personnes qui ont contribué, de près ou de loin, à la réalisation de ce projet.

Enfin, nous adressons nos sincères remerciements aux membres du jury pour l'intérêt qu'ils ont porté à notre travail, ainsi que pour leurs remarques et suggestions constructives qui contribueront à enrichir notre réflexion et à améliorer ce travail.

Résumé

Le diagnostic automobile moderne repose sur l'exploitation des données issues du bus de communication interne du véhicule, notamment le bus Controller Area Network (CAN), accessible via l'interface normalisée OBD-II. Toutefois, les solutions existantes se répartissent principalement entre des outils professionnels offrant un accès avancé mais caractérisés par un coût élevé et une architecture fermée, et des dispositifs grand public plus accessibles mais limités en termes de performances et de fonctionnalités.

Dans ce contexte, ce travail propose la conception et la réalisation d'un dispositif embarqué dédié à l'acquisition fiable des trames CAN à partir du port OBD-II. Le système développé repose sur l'utilisation d'un contrôleur CAN externe, associé à une unité de traitement assurant la gestion des protocoles de communication, l'implémentation des services OBD-II et la reconstruction des messages multi-trames conformément aux spécifications du protocole ISO-TP.

L'architecture proposée permet une capture exhaustive des échanges sur le bus CAN, ainsi qu'une structuration des données sous forme de flux exploitable par des systèmes externes. Une interface locale de visualisation, basée sur une plateforme embarquée, permet également l'affichage en temps réel de certaines informations diagnostiques.

La validation expérimentale a été réalisée sur plusieurs véhicules présentant des configurations hétérogènes. Les résultats obtenus montrent une amélioration significative en termes de fiabilité d'acquisition, notamment pour l'écoute passive du bus et la gestion des échanges multi-trames, confirmant ainsi la pertinence de l'approche proposée.

Ce dispositif s'inscrit dans une architecture plus large de diagnostic automobile, où les données acquises peuvent être exploitées par des systèmes externes d'analyse avancée.

Mots-clés : bus CAN, interface OBD-II, acquisition de trames CAN, système embarqué, contrôleur CAN, ISO-TP, diagnostic embarqué.

Abstract

Modern automotive diagnostics relies on the exploitation of data generated by the vehicle's internal communication network, particularly the Controller Area Network (CAN) bus, accessible through the standardized OBD-II interface. However, existing solutions are mainly divided between professional tools that provide advanced access but are characterized by high cost and closed architectures, and consumer-grade devices that are more accessible but limited in terms of performance and functionality.

In this context, this work proposes the design and realization of an embedded device dedicated to the reliable acquisition of CAN frames from the OBD-II port. The developed system is based on the use of an external CAN controller, coupled with a processing unit responsible for managing communication protocols, implementing OBD-II services, and reconstructing multi-frame messages in accordance with ISO-TP specifications.

The proposed architecture enables comprehensive capture of CAN bus traffic, as well as the structuring of the data into a stream that can be exploited by external systems. A local visualization interface, based on an embedded platform, also allows real-time display of selected diagnostic information.

Experimental validation was carried out on multiple vehicles with heterogeneous configurations. The obtained results demonstrate a significant improvement in acquisition reliability, particularly for passive bus monitoring and multi-frame message handling, thereby confirming the relevance of the proposed approach.

This device is part of a broader automotive diagnostic architecture, in which the acquired data can be utilized by external advanced analysis systems.

Keywords : CAN bus, OBD-II interface, CAN frame acquisition, embedded system, CAN controller, ISO-TP, automotive diagnostics.

الملخص

يعتمد التشخيص الحديث للمركبات على استغلال البيانات الصادرة عن شبكة الاتصال الداخلية للمركبة، وخاصة ناقل Controller Area Network (CAN) ، والذي يمكن الوصول إليه عبر واجهة OBD-II المعيارية. ومع ذلك، تنقسم الحلول المتوفرة حالياً بشكل رئيسي إلى أدوات احترافية توفر وصولاً متقدماً لكنها تتميز بتكلفة مرتفعة وبنية مغلقة، وأجهزة موجهة للمستهلك أكثر سهولة في الاستخدام ولكنها محدودة من حيث الأداء والوظائف.

في هذا السياق، يقترح هذا العمل تصميم وإنجاز جهاز مدمج مخصص لاكتساب إطارات CAN بشكل موثوق انطلاقاً من منفذ OBD-II . يعتمد النظام المطور على استخدام متحكم CAN خارجي مقترن بوحدة معالجة تتكفل بإدارة بروتوكولات الاتصال، وتنفيذ خدمات OBD-II ، وإعادة تركيب الرسائل متعددة الإطارات وفقاً لمواصفات بروتوكول ISO-TP .

تتيح البنية المقترحة التقاطاً شاملاً لحركة البيانات على ناقل CAN ، إضافة إلى تنظيم هذه البيانات في شكل تدفق قابل للاستغلال من قبل أنظمة خارجية. كما تتيح واجهة عرض محلية، مبنية على منصة مدمجة، عرض بعض المعلومات التشخيصية بشكل آني.

تم إجراء التحقق التجريبي على عدة مركبات ذات خصائص مختلفة. وقد أظهرت النتائج تحسناً ملحوظاً في موثوقية الاكتساب، خاصة فيما يتعلق بالمراقبة السلبية للناقل ومعالجة الرسائل متعددة الإطارات، مما يؤكد فعالية النهج المقترح.

يندرج هذا الجهاز ضمن بنية أوسع للتشخيص الآلي للمركبات، حيث يمكن استغلال البيانات المكتسبة من قبل أنظمة تحليل متقدمة خارجية.

الكلمات المفتاحية : ناقل CAN ، واجهة OBD-II ، التشخيص الإلكتروني للسيارات، الجهاز المدمج ، اقتناء البيانات، ISO-TP ، المتحكم الدقيق.

Table des matières

Résumés **6**

Liste des figures **12**

Liste des tableaux **13**

Liste des abréviations **14**

Introduction **16**
1 Contexte général 17
2 Problématique 17
3 Objectifs du travail 18
4 Organisation du mémoire 18

1 État de l'art du diagnostic automobile et de l'acquisition de données sur bus CAN **20**
1.1 Introduction 20
1.2 Évolution du diagnostic automobile 20
1.3 Typologie des outils de diagnostic actuels 22
1.3.1 Les valises professionnelles multi-marques 22
1.3.2 Les adaptateurs ELM327 et dérivés grand public 22
1.3.3 Les plateformes ouvertes, open-source et commerciales d'acquisition CAN 23
1.4 Travaux académiques sur l'acquisition CAN et le diagnostic embarqué 24
1.4.1 Architectures d'acquisition CAN 24
1.4.2 Diagnostic OBD-II connecté 24
1.4.3 Tableaux de bord embarqués fondés sur CAN 25
1.4.4 Sécurité automobile et rétro-ingénierie CAN 25
1.5 Limites identifiées dans la littérature et les solutions existantes 26
1.6 Positionnement et contribution de ce travail 28
1.7 Conclusion 28

2 Fondements théoriques et technologies du système VEYA **30**
2.1 Introduction 30
2.2 Architecture électronique des véhicules modernes 30
2.3 Interface OBD-II et accès au diagnostic 31

2.4 Bus CAN : principes, trames et couche physique 32
2.5 Transport ISO-TP sur CAN 33
2.6 Services UDS pour le diagnostic 34
2.6.1 Principe général des services UDS 35
2.6.2 Services UDS utiles au diagnostic 36
2.6.3 Notion de code défaut 36
2.7 KWP2000 et communication K-Line 36
2.8 Architecture matérielle utilisée dans le projet 37
2.8.1 Adaptateurs ELM327 et premières expérimentations 38
2.8.2 OBDLink SX comme interface de communication avancée . . 39
2.8.3 Famille ESP32 comme cible embarquée 40
2.8.4 Contrôleur CAN MCP2515 41
2.8.5 Transcepteur CAN 42
2.8.6 Interface K-Line 42
2.8.7 Raspberry Pi 5 et écran embarqué 43
2.9 Environnement logiciel et outils de développement 44
2.9.1 Python et PySerial 44
2.9.2 SQLite et base DDT2000 44
2.9.3 Wireshark et USBPcap 45
2.9.4 Qt/QML, C++ et JavaScript 45
2.9.5 C/C++ pour le firmware embarqué 45
2.9.6 CMake, Bash, Git et GitHub 45
2.10 Protocoles et formats de communication logiciels 46
2.10.1 JSON 46
2.10.2 TCP 46
2.10.3 WebSocket 47
2.11 Conclusion 47

3 Conception du système embarqué VEYA **48**
3.1 Introduction 48
3.2 Besoins fonctionnels et objectifs du système 48
3.2.1 Fonctions principales 48
3.2.2 Diagnostic distant connecté 49
3.2.3 Objectifs de conception 49
3.3 Vue générale de l'architecture VEYA 49
3.4 Conception du tableau de bord embarqué 50
3.4.1 Rôle du tableau de bord embarqué 50
3.4.2 Organisation logicielle de l'interface 50
3.4.3 Profils et écrans fonctionnels 51
3.4.4 Gestion des sources de données 52
3.4.5 Intégration mécanique et ergonomie 53
3.5 Conception du dispositif d'acquisition OBD 54
3.5.1 Rôle du dispositif d'acquisition 54
3.5.2 Architecture matérielle retenue 54
3.5.3 Interfaces véhicule et acquisition CAN 55
3.5.4 Alimentation et protection électrique 56
3.5.5 Sélection de mode et voyants 57
3.5.6 Communication et format des données 57

3.5.7 Intégration mécanique du boîtier 58
3.6 Choix de la communication entre dispositif OBD et Raspberry Pi . . 58
3.6.1 Besoin d'une liaison sans fil locale 58
3.6.2 Justification du choix BLE 59
3.6.3 Rôle du pont BLE sur Raspberry Pi 59
3.6.4 Modèle de messages échangés 60
3.6.5 Considérations de fiabilité 60
3.7 Conception de la chaîne d'acquisition et de diagnostic 61
3.7.1 Nature des données acquises 61
3.7.2 Chaîne locale d'acquisition et de diffusion 61
3.7.3 Machine d'états de sélection de source 61
3.7.4 Flux des commandes de diagnostic 62
3.7.5 Couche distante de rapport et session live 62
3.7.6 Enrichissement constructeur Renault/Dacia par XML DDT2000 62
3.7.7 Encadrement des opérations de diagnostic 63
3.8 Conclusion 63

4 Mise en œuvre expérimentale et validation 64
4.1 Introduction 64
4.2 Mise en œuvre expérimentale de la chaîne de diagnostic 64
4.3 Premiers essais avec adaptateurs ELM327 Bluetooth 65
4.3.1 Objectif de la phase ELM327 65
4.3.2 Véhicules et conditions d'essai 65
4.3.3 Résultats par véhicule 66
4.3.4 Synthèse des limites observées 67
4.3.5 Apports pour l'architecture VEYA 67
4.4 Transition vers des interfaces plus performantes 67
4.4.1 Limites observées avec les interfaces Bluetooth génériques . . 67
4.4.2 Intérêt des interfaces USB et de l'OBDLink SX 68
4.4.3 Conséquences sur la méthode de développement 68
4.5 Analyse approfondie des protocoles et outils Renault/Dacia 68
4.5.1 Complémentarité des outils spécialisés 68
4.5.2 Capture et analyse des échanges 69
4.5.3 Exploitation des fichiers XML DDT2000 69
4.5.4 Positionnement des protocoles 69
4.5.5 Chaîne de traitement expérimentale 70
4.5.6 Encadrement des opérations et discipline de preuve 70
4.6 Résultats obtenus sur Renault Symbol 2019 70
4.6.1 Objectif de l'essai sur Renault Symbol 2019 70
4.6.2 Identification des contextes ECU 71
4.6.3 Extraction des DTC et interprétation 71
4.6.4 Extraction du kilométrage 72
4.6.5 Apport de l'enrichissement DDT2000 72
4.6.6 Bilan de validation 72
4.7 Résultats obtenus sur Dacia Duster 2012 72
4.7.1 Objectif de l'essai sur Dacia Duster 2012 72
4.7.2 Mise en évidence du contexte KWP2000/K-Line 73
4.7.3 Calculateurs répondants et adresses identifiées 73

4.7.4 Requêtes d'identification et observations `21xx` 74
4.7.5 Évolution de la lecture DTC et limites initiales 74
4.7.6 Recherche des fichiers XML candidats 75
4.7.7 Bilan de validation et impact sur VEYA 75
4.8 Intégration de la chaîne locale VEYA 76
4.8.1 Objectif de l'intégration locale 76
4.8.2 Validation de la liaison BLE ESP32-S3 / Raspberry Pi 76
4.8.3 Organisation logicielle sur Raspberry Pi 76
4.8.4 Intégration avec le tableau de bord 77
4.8.5 Flux des commandes de diagnostic 77
4.8.6 Liaison avec le service distant 78
4.8.7 Bilan d'intégration 78
4.9 Réalisation du prototype matériel retenu 78
4.9.1 Objectif du prototype matériel 78
4.9.2 Composition matérielle du dispositif 79
4.9.3 Réalisation de l'alimentation et des protections 81
4.9.4 Sélection de mode et indicateurs 81
4.9.5 Développement du firmware ESP32-S3 82
4.9.6 État de validation du prototype 82
4.9.7 Intégration mécanique et boîtier 82
4.10 Validation globale et discussion 83
4.10.1 Synthèse de la validation globale 83
4.10.2 Niveaux de validation obtenus 83
4.10.3 Discussion des choix expérimentaux 83
4.10.4 Limites et perspectives 84
4.11 Conclusion 85

Conclusion générale **86**

Conclusion générale **86**

Bibliographie **88**

A Organisation logicielle du tableau de bord VEYA **91**
A.1 Présentation générale 91
A.2 Arborescence simplifiée du projet 91
A.3 Couche de services Python 92
A.4 Interface graphique Qt 6/QML 92
A.5 Accès au dépôt du projet 93

B Organisation du micrologiciel ESP32-S3 sous ESP-IDF **94**
B.1 Présentation du projet ESP-IDF 94
B.2 Arborescence du micrologiciel 94
B.3 Fichiers de compilation et de configuration 95
B.4 Organisation fonctionnelle du programme principal 95
B.5 Dépendances logicielles 95

C Dossier du projet “Diplôme/Startup” dans le cadre de l’Arrêté Ministériel N° 1275 **96**

Liste des figures

Fig. 1.1 Architecture simplifiée d'un réseau automobile composé de plusieurs segments CAN interconnectés par une passerelle. . . . 21
Fig. 1.2 Évolution synthétique du diagnostic automobile vers les architectures CAN/OBD-II modernes. 22

Fig. 2.1 Architecture électronique simplifiée d'un véhicule moderne intégrant plusieurs UCE, réseaux embarqués, passerelle et accès diagnostic. 31
Fig. 2.2 Brochage du connecteur OBD-II et signaux utilisés pour l'accès au diagnostic. 32
Fig. 2.3 Structure simplifiée d'une trame CAN classique. 33
Fig. 2.4 Chaîne fonctionnelle entre l'unité de traitement, le contrôleur CAN, le transcepteur et le bus véhicule. 33
Fig. 2.5 Principe d'un échange ISO-TP multi-trame sur CAN. 34
Fig. 2.6 Modèle client-serveur des services UDS. 35
Fig. 2.7 Chaîne matérielle générique d'un dispositif d'acquisition CAN/K-Line. 38
Fig. 2.8 Exemple d'architecture détaillée d'un adaptateur de diagnostic fondé sur l'ELM327. 39
Fig. 2.9 Adaptateur OBDLink SX utilisé comme interface de communication avancée. 40
Fig. 2.10 Schéma fonctionnel simplifié de la famille ESP32 comme cible embarquée. 41
Fig. 2.11 Schéma fonctionnel du contrôleur CAN MCP2515. 41
Fig. 2.12 Schéma fonctionnel d'un transcepteur CAN. 42
Fig. 2.13 Schéma fonctionnel du circuit L9637D pour l'interface K-Line. 43
Fig. 2.14 Carte Raspberry Pi 5 envisagée pour l'interface embarquée. . . 43
Fig. 2.15 Positionnement des formats et protocoles logiciels dans une chaîne d'échange de données. 46

Fig. 3.1 Architecture globale de VEYA : chaîne embarquée locale et service distant de diagnostic. 50
Fig. 3.2 Écrans principaux de l'interface VEYA : accueil, conduite et diagnostic. 52
Fig. 3.3 Écrans de configuration de la connectivité du tableau de bord VEYA : Wi-Fi et Bluetooth. 52
Fig. 3.4 Rendus SolidWorks du boîtier du tableau de bord embarqué VEYA. 53
Fig. 3.5 Boîtier imprimé du tableau de bord embarqué VEYA. 54

Fig. 3.6 Repérage des broches de la carte ESP32-S3 utilisée dans le prototype VEYA. 55
Fig. 3.7 Architecture fonctionnelle du dispositif d'acquisition OBD retenu pour VEYA. 55
Fig. 3.8 Brochage de référence de l'adaptateur OBD-II vers DB9 utilisé pour le raccordement véhicule du dispositif VEYA. 56
Fig. 3.9 Schéma fonctionnel de câblage du dispositif d'acquisition OBD VEYA, montrant la chaîne d'alimentation protégée, l'interface USB OBDLink SX, l'interface CAN via MCP2515 et les signaux de contrôle/état de l'ESP32-S3. 57
Fig. 3.10 Boîtier imprimé du dispositif d'acquisition OBD VEYA. 58
Fig. 3.11 Flux bidirectionnels entre le dispositif d'acquisition BLE et le tableau de bord Raspberry Pi. 59
Fig. 3.12 Chaîne locale d'acquisition et flux de diagnostic de VEYA, complétés par la couche distante externe. 61

Fig. 4.1 Exemples de véhicules utilisés lors de la phase initiale d'essais avec adaptateur ELM327 Bluetooth : Renault Symbol 2010, Citroën Berlingo 2016 et Hyundai Accent 2020. 65
Fig. 4.2 Chaîne expérimentale d'analyse des outils et protocoles Renault/Dacia. 70
Fig. 4.3 Contexte expérimental sur Dacia Duster 2012 : compartiment moteur et montage d'essai. 73
Fig. 4.4 Intégration de la chaîne locale VEYA et raccordement au service distant de diagnostic. 77
Fig. 4.5 Prototype assemblé du dispositif d'acquisition OBD VEYA : vue avant et vue arrière. 81

Liste des tableaux

1.1 Typologie des solutions de diagnostic et d'acquisition automobile. . . 24
1.2 Comparaison des travaux académiques liés à l'acquisition CAN et au diagnostic embarqué. 26
1.3 Synthèse des limites identifiées et des exigences associées au dispositif proposé. 27

2.1 Broches OBD-II pertinentes pour le dispositif d'acquisition. 32
2.2 Rôle des principaux champs d'une trame CAN classique. 33
2.3 Types de trames définis par ISO-TP. 34
2.4 Principaux services UDS normalisés par l'ISO 14229. 35
2.5 Comparaison synthétique entre diagnostic sur CAN et communication K-Line/KWP2000. 37
2.6 Évolution des interfaces d'acquisition utilisées ou envisagées dans le projet. 38

2.7 Environnement logiciel et outils utilisés dans le projet. 44
2.8 Comparaison des formats et protocoles logiciels utilisés pour l'échange des données. 46

4.1 Synthèse des premiers essais réalisés avec un adaptateur ELM327 Bluetooth clone. 67
4.2 Contextes ECU exploités pour la Renault Symbol 2019 71
4.3 DTC extraits de la capture PCAP V2 de la Renault Symbol 2019 71
4.4 Calculateurs répondants identifiés sur la Dacia Duster 2012 74
4.5 Fichiers XML candidats utilisés pour le sondage de la Dacia Duster 2012 75
4.6 Blocs du prototype matériel et niveau de validation documenté. . . . 80
4.7 Synthèse des niveaux de validation obtenus pour les blocs de VEYA. 83

Liste des abréviations

Acronyme	Signification
API	Application Programming Interface, interface de programmation applicative
BFR	Besoin en fonds de roulement
BLE	Bluetooth Low Energy
BMC	Business Model Canvas
CAN	Controller Area Network
CAN FD	Controller Area Network Flexible Data-rate
CDC-ACM	Communications Device Class, Abstract Control Model
DID	Data Identifier
DTC	Diagnostic Trouble Code, code de défaut diagnostic
ECU	Electronic Control Unit, calculateur électronique
ESSAT	École Supérieure des Sciences Appliquées de Tlemcen
GPIO	General-Purpose Input/Output
HMI / IHM	Human–Machine Interface / Interface Homme–Machine
HTTP / HTTPS	Hypertext Transfer Protocol / Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
ISO	International Organization for Standardization
ISO-TP	ISO Transport Protocol
JSON	JavaScript Object Notation
KWP2000	Keyword Protocol 2000
MCU	Microcontroller Unit
MIL	Malfunction Indicator Lamp, voyant de défaut moteur
NIV / VIN	Numéro d'identification du véhicule / Vehicle Identification Number
OBD	On-Board Diagnostics
OBD-II	On-Board Diagnostics, second generation
PCB	Printed Circuit Board, carte de circuit imprimé
PDF	Portable Document Format
PID	Parameter Identifier
QML	Qt Modeling Language
RPM	Revolutions Per Minute, tours par minute
RTOS	Real-Time Operating System
SAE	Society of Automotive Engineers

Acronyme	Signification
SPI	Serial Peripheral Interface
SQL	Structured Query Language
TCP	Transmission Control Protocol
UART	Universal Asynchronous Receiver-Transmitter
UDS	Unified Diagnostic Services
USB	Universal Serial Bus
UUID	Universally Unique Identifier
VCP	Virtual COM Port
VEYA	Vehicle Eye Analysis
WebSocket	Protocole de communication bidirectionnelle sur TCP
XML	Extensible Markup Language

Introduction générale

1 Contexte général

L'automobile contemporaine est devenue un système cyber-physique fortement intégré, dans lequel le fonctionnement mécanique est piloté, surveillé et optimisé par un réseau d'unités de contrôle électronique (UCE) interconnectées [20]. Un véhicule de génération récente embarque couramment plusieurs dizaines de ces calculateurs, chacun dédié à une fonction spécifique : gestion moteur, transmission, freinage, airbag, climatisation ou habitacle. Leur coordination repose sur des bus de communication série, parmi lesquels le bus CAN (*Controller Area Network*), introduit par Bosch en 1983 puis normalisé sous la référence ISO 11898, s'est imposé comme l'épine dorsale de l'électronique embarquée automobile [24, 9, 27].

Parallèlement, la pression réglementaire en matière d'émissions polluantes a conduit à la standardisation du diagnostic embarqué OBD-II, dont le connecteur universel à seize broches défini par la norme SAE J1962 et les modes de service décrits par la norme SAE J1979 constituent un point d'accès normalisé aux données des UCE [25, 26, 10, 17]. Cette interface, obligatoire sur tous les véhicules commercialisés aux États-Unis depuis 1996 puis en Europe depuis 2001 pour les motorisations essence, est devenue la porte d'entrée universelle de tout outil de diagnostic, qu'il soit professionnel ou grand public.

Dans le contexte algérien, le parc automobile demeure majoritairement composé de véhicules âgés de plus de dix ans, pour lesquels les outils de diagnostic propriétaires des constructeurs sont difficilement accessibles, tant par leur coût que par la restriction de leur distribution. Les utilisateurs et les réparateurs indépendants se reportent sur des adaptateurs OBD-II génériques, dont les limites fonctionnelles sont rapidement atteintes dès que la démarche de diagnostic sort du périmètre strictement OBD [29].

2 Problématique

Les outils de diagnostic présents sur le marché se répartissent en deux catégories aux caractéristiques opposées. D'une part, les valises professionnelles multi-marques offrent un accès étendu aux calculateurs mais demeurent propriétaires, fermées et coûteuses, ce qui les rend peu compatibles avec un usage généralisé dans un tissu de réparation indépendant. D'autre part, les adaptateurs OBD-II génériques, en particulier les clones de la puce ELM327, sont largement accessibles mais souffrent de limitations structurelles : gestion imparfaite du transport multi-trame ISO-TP,

absence d'accès aux codes défauts spécifiques au constructeur, capture passive du bus CAN bridée par le micrologiciel, incapacité à dialoguer de manière fiable avec les calculateurs situés hors du groupe motopropulseur. Ces limitations ont été confirmées de manière empirique lors des essais préliminaires conduits dans le cadre du présent projet sur plusieurs véhicules représentatifs du parc circulant.

Le besoin identifié est celui d'un dispositif d'acquisition ouvert, maîtrisé et extensible, capable de capturer les trames du bus CAN à pleine vitesse, de dialoguer avec l'ensemble des calculateurs accessibles à travers l'interface OBD-II, et de servir de socle matériel à une plateforme complète de diagnostic. Un tel dispositif doit en outre rester économiquement accessible afin de répondre aux contraintes du marché local.

3 Objectifs du travail

Le présent mémoire s'inscrit dans le cadre du projet **VEYA** (*Vehicle Eye Analysis*), qui vise à développer une plateforme embarquée complète de diagnostic automobile. Ce travail traite spécifiquement de la phase d'ingénierie du projet, consacrée à la conception et à la réalisation du dispositif d'acquisition des trames du bus CAN ; la phase logicielle de visualisation et d'analyse, menée en parallèle dans le cadre de la formation de Master, ne relève pas du périmètre du présent manuscrit mais en constitue le prolongement naturel.

Les objectifs poursuivis sont les suivants. Il s'agit tout d'abord d'étudier les architectures matérielles et logicielles existantes pour l'acquisition de données sur bus CAN via l'interface OBD-II, afin d'identifier les choix technologiques pertinents et les écueils documentés [22, 1, 28, 14]. Il s'agit ensuite de concevoir un dispositif embarqué reposant sur un microcontrôleur associé à un contrôleur CAN dédié, offrant un accès direct au bus, une latence réduite et une capacité de sniffing passif à pleine vitesse — caractéristiques qui font défaut aux adaptateurs ELM327 clones. Le travail comporte également le développement du micrologiciel associé, assurant l'initialisation du contrôleur CAN, la lecture et le filtrage des trames, la gestion du transport ISO-TP multi-trame, ainsi que le dialogue avec les calculateurs à travers les services OBD standard et, lorsque cela s'avère pertinent, les services de diagnostic unifié (UDS). Le dispositif est enfin validé sur véhicules réels, par comparaison avec un outil de référence professionnel et sur une base empirique couvrant plusieurs constructeurs et motorisations, afin de caractériser objectivement ses performances et ses limites. L'ensemble de la démarche est documenté sous une forme reproductible, de sorte que le dispositif puisse être intégré ultérieurement à la plateforme logicielle développée dans la phase Master du projet.

4 Organisation du mémoire

Le manuscrit est structuré en quatre chapitres, encadrés par la présente introduction et par une conclusion générale.

Le chapitre 1 dresse l'état de l'art des systèmes de diagnostic automobile et des dispositifs embarqués d'acquisition de données CAN. Les travaux académiques et les solutions commerciales y sont analysés, et le positionnement du présent travail y est établi.

Le chapitre 2 présente l'étude théorique sur laquelle s'appuie la conception. Y sont successivement abordés le bus CAN (couches physique et liaison, normalisation), le protocole OBD-II (connecteur, modes de service, protocoles de transport sous-jacents), l'architecture des unités de contrôle électronique et les principes généraux des systèmes embarqués temps réel.

Le chapitre 3 expose la conception et la réalisation du dispositif. Il y est décrit l'architecture globale du système, les choix matériels retenus, la conception électronique, le développement du micrologiciel et les protocoles implémentés.

Le chapitre 4 est consacré aux résultats expérimentaux. Les campagnes de test menées sur plusieurs véhicules représentatifs y sont analysées, une comparaison avec un outil de diagnostic professionnel de référence y est établie, et les performances du dispositif y sont caractérisées.

La conclusion générale synthétise les apports du travail, en dégage les limites et propose des perspectives d'évolution, notamment l'intégration du dispositif à la plateforme VEYA et l'extension de ses capacités vers des fonctions avancées de diagnostic.

CHAPITRE 1

État de l'art du diagnostic automobile et de l'acquisition de données sur bus CAN

1.1 Introduction

Ce chapitre présente l'état de l'art du diagnostic automobile et des dispositifs d'acquisition de données sur bus CAN. Il vise à situer le présent travail dans l'évolution des architectures électroniques des véhicules, des outils de diagnostic et des solutions d'acquisition existantes. L'objectif est d'identifier les limites des approches actuelles afin de justifier la conception d'un dispositif embarqué ouvert, évolutif et adapté à l'acquisition de trames CAN via l'interface OBD-II.

1.2 Évolution du diagnostic automobile

Le diagnostic automobile a évolué parallèlement à l'augmentation de la part de l'électronique dans les véhicules. Les fonctions mécaniques, autrefois principalement analysées par observation directe et mesures ponctuelles, sont aujourd'hui pilotées et surveillées par des unités de contrôle électronique interconnectées. Ces calculateurs gèrent des fonctions variées telles que l'injection, l'allumage, la transmission, le freinage, la sécurité passive, le confort et l'habitacle [20, 23]. Cette évolution a transformé le diagnostic en une activité fondée sur l'accès aux données internes du véhicule.

La multiplication des calculateurs a rendu nécessaire l'utilisation de réseaux de communication embarqués. Le bus CAN (*Controller Area Network*) s'est imposé comme l'une des solutions les plus utilisées dans l'automobile en raison de son fonctionnement multiplexé, de son mécanisme d'arbitrage et de sa robustesse en environnement perturbé [24, 9, 27]. Au lieu de relier chaque fonction par un câblage point à point, le CAN permet à plusieurs nœuds d'échanger des messages sur un support partagé. Cette architecture réduit la complexité du faisceau électrique et facilite l'accès centralisé à certaines informations du véhicule.

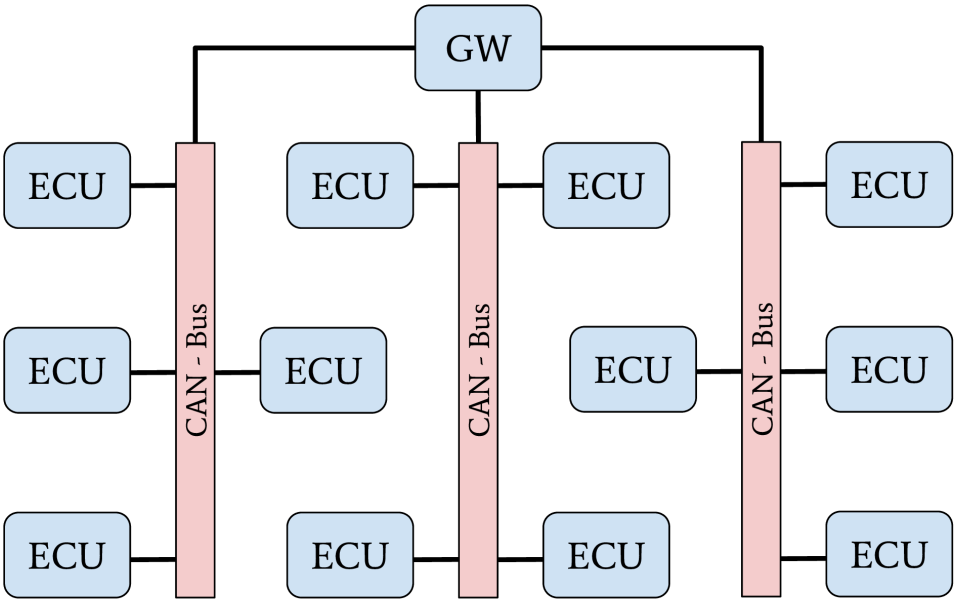


FIGURE 1.1 – Architecture simplifiée d'un réseau automobile composé de plusieurs segments CAN interconnectés par une passerelle.

L'apparition du diagnostic embarqué normalisé a ensuite donné un cadre commun à l'interrogation des véhicules. L'interface OBD-II définit un point d'accès physique au véhicule à travers un connecteur normalisé, tandis que les services de diagnostic associés permettent de lire des paramètres, des codes défauts et certaines informations d'identification [25, 26, 10]. Dans de nombreux véhicules récents, ces échanges reposent sur des protocoles de diagnostic transportés sur CAN, notamment à travers des mécanismes de transport multi-trame définis par la famille ISO 15765 [12]. Ainsi, le diagnostic moderne ne dépend pas seulement de la présence d'un connecteur OBD-II, mais aussi de la capacité de l'outil à gérer correctement les couches de communication sous-jacentes.

Cette évolution a également introduit des protocoles de diagnostic plus avancés. Les services de diagnostic unifiés, définis par l'ISO 14229, permettent d'accéder à des fonctions plus étendues que celles du diagnostic OBD-II standard, notamment la lecture d'identifiants de données, la gestion de sessions de diagnostic ou la lecture d'informations de défaut plus détaillées [13]. De même, certains véhicules plus anciens utilisent encore des protocoles antérieurs au diagnostic CAN généralisé, tels que KWP2000 sur ligne K, défini par l'ISO 14230 [11]. Un système d'acquisition réellement exploitable doit donc tenir compte de l'hétérogénéité des véhicules, des supports physiques et des couches protocolaires.

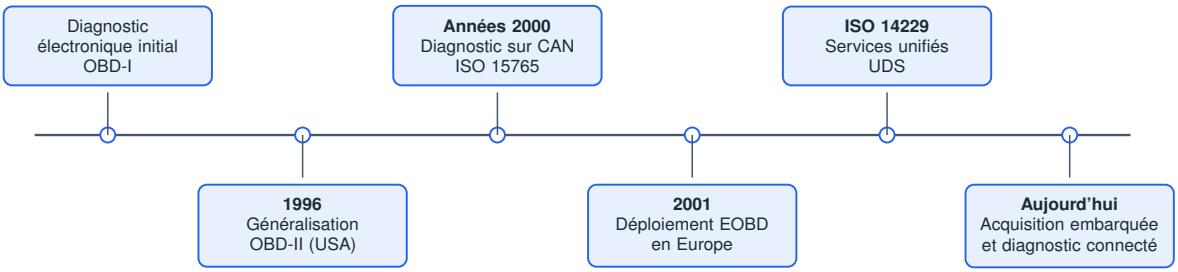


FIGURE 1.2 – Évolution synthétique du diagnostic automobile vers les architectures CAN/OBD-II modernes.

1.3 Typologie des outils de diagnostic actuels

1.3.1 Les valises professionnelles multi-marques

Les valises professionnelles multi-marques constituent la catégorie d'outils la plus complète pour les ateliers spécialisés. Elles sont conçues pour accéder à plusieurs calculateurs, lire les codes défauts, afficher des paramètres en temps réel, lancer des procédures de test et parfois effectuer des opérations de maintenance ou de configuration. Leur intérêt principal réside dans la couverture fonctionnelle offerte à l'utilisateur final, en particulier lorsque l'objectif est de réaliser une intervention rapide et guidée sur un véhicule.

Cependant, ces outils sont généralement pensés comme des systèmes intégrés et fermés. L'utilisateur interagit avec une interface applicative, mais ne dispose pas nécessairement d'un accès direct aux trames échangées sur le bus, aux détails du transport multi-trame ou aux mécanismes internes d'interrogation des calculateurs. Pour une démarche d'ingénierie ou de recherche, cette abstraction constitue une limite, car elle empêche de maîtriser entièrement la chaîne d'acquisition, depuis la trame brute jusqu'à la donnée interprétée. L'enjeu du présent travail n'est donc pas de reproduire une valise professionnelle complète, mais de construire une couche d'acquisition maîtrisée, capable d'alimenter ensuite des traitements de diagnostic.

1.3.2 Les adaptateurs ELM327 et dérivés grand public

Les adaptateurs fondés sur l'architecture ELM327 représentent une autre catégorie très répandue. Le composant ELM327 est conçu comme un interpréteur OBD qui fournit une interface à commandes de type AT entre un hôte et les protocoles de diagnostic du véhicule [5]. Cette approche a largement contribué à démocratiser l'accès aux données OBD-II, car elle permet à un ordinateur ou à un téléphone d'interroger facilement des paramètres standardisés sans gérer directement les détails bas niveau de la communication véhicule.

Ce modèle présente néanmoins une limite importante pour l'acquisition de données. L'adaptateur masque une partie de la complexité protocolaire derrière une interface de haut niveau. Cette abstraction est suffisante pour lire des paramètres OBD-II courants, mais elle devient moins adaptée lorsque l'objectif est de capturer des trames CAN de manière exhaustive, d'horodater précisément les échanges, d'observer passivement le bus ou de maîtriser finement la gestion du transport multi-trame. Dans une architecture de recherche ou de développement, le concepteur doit accéder

directement à la couche CAN afin de contrôler la réception, le filtrage, la transmission et le réassemblage des messages.

Les adaptateurs grand public conservent donc un intérêt pour le diagnostic simple, mais ils ne constituent pas une base idéale pour développer un dispositif d'acquisition évolutif. Leur rôle est davantage celui d'une passerelle pratique vers les services OBD-II standard que celui d'un contrôleur d'acquisition transparent et programmable.

1.3.3 Les plateformes ouvertes, open-source et commerciales d'acquisition CAN

Entre les valises professionnelles et les adaptateurs grand public, il existe des plateformes davantage orientées vers l'acquisition de trames CAN. Certaines sont ouvertes ou semi-ouvertes, tandis que d'autres prennent la forme d'enregistreurs commerciaux spécialisés. Le projet *rpi-can-logger*, par exemple, illustre une approche fondée sur un Raspberry Pi associé à une interface CAN afin d'enregistrer des trames et de les exploiter dans un environnement logiciel modifiable [29]. Ce type d'architecture présente l'intérêt de rapprocher l'utilisateur de la donnée brute et de favoriser la reproductibilité.

L'écosystème logiciel joue également un rôle important dans ces approches. Des outils comme SocketCAN, intégré à l'environnement Linux, ou des bibliothèques telles que *python-can* et *python-isotp*, permettent de construire des chaînes de traitement où la réception des trames, l'émission de requêtes et la gestion du transport ISO-TP peuvent être contrôlées par logiciel [7, 6, 16]. Ces solutions montrent que l'acquisition CAN ne se limite pas au matériel : elle nécessite aussi une couche logicielle capable de structurer, filtrer et transmettre les données vers des modules de traitement.

Les enregistreurs CAN commerciaux illustrent pour leur part les fonctionnalités attendues d'une chaîne d'acquisition robuste. Les produits présentés dans le catalogue CSS Electronics, tels que les familles CLX000 et CANedge, combinent généralement l'enregistrement horodaté, le stockage local, le filtrage, l'export de données et, dans certaines versions, la transmission vers un serveur [4]. Ces solutions confirment l'importance de fonctions comme l'horodatage, le filtrage, la journalisation et l'interopérabilité avec des outils logiciels. Toutefois, elles restent des produits prêts à l'emploi, alors que le présent travail vise la conception d'une architecture académique maîtrisée, spécifiquement orientée vers l'interface OBD-II et l'intégration avec une chaîne de diagnostic automobile.

Tableau 1.1 – Typologie des solutions de diagnostic et d'acquisition automobile.

Catégorie	Forces principales	Limites principales	Exemples
Valises professionnelles	Accès fonctionnel avancé, couverture multi-calculateurs, procédures guidées	Architecture fermée, coût élevé, accès limité aux trames brutes	Outils constructeurs, valises multi-marques
Adaptateurs ELM327	Faible coût, simplicité d'utilisation, accès aux services OBD-II standard	Interface de haut niveau, maîtrise réduite du sniffing et du transport multi-trame	Adaptateurs USB, Bluetooth ou Wi-Fi
Plateformes ouvertes et loggers CAN	Accès aux trames, reproductibilité, intégration avec outils logiciels	Nécessitent une configuration technique et une couche de décodage	Raspberry Pi + interface CAN, SocketCAN, enregistreurs CAN

1.4 Travaux académiques sur l'acquisition CAN et le diagnostic embarqué

1.4.1 Architectures d'acquisition CAN

Plusieurs travaux académiques se sont intéressés à l'acquisition de données sur bus véhicule. Wang et al. proposent un système d'acquisition de données de bus automobile associé à une fonction de diagnostic de défauts [28]. Ce type d'architecture met en évidence l'intérêt d'une couche d'acquisition capable de collecter les informations provenant du réseau du véhicule avant leur exploitation logicielle.

Chen et Li présentent un système d'acquisition et de surveillance de données CAN fondé sur LabVIEW [1]. Cette approche met l'accent sur la supervision, l'affichage et l'analyse dans un environnement logiciel de laboratoire. Elle convient particulièrement aux essais expérimentaux et aux bancs de test, où l'ordinateur hôte joue un rôle central. En revanche, elle est moins proche d'un dispositif embarqué autonome, compact et directement intégrable dans un véhicule. Cette distinction est importante pour le présent travail, qui cherche à concevoir une couche d'acquisition embarquée plutôt qu'un système de supervision dépendant d'un poste de laboratoire.

Ces travaux montrent que l'acquisition CAN est un élément récurrent dans les architectures de diagnostic ou de surveillance, avec des choix dépendant du contexte d'utilisation.

1.4.2 Diagnostic OBD-II connecté

L'exploitation de l'OBD-II ne se limite pas à la lecture locale de paramètres. Jhou et al. proposent une architecture de diagnostic OBD-II intégrée à une infrastructure de calcul distante [14]. Dans ce type de système, les données extraites du véhicule sont transmises vers une couche externe, où elles peuvent être stockées, consultées ou analysées. L'intérêt de cette approche réside dans la séparation entre l'acquisition embarquée et le traitement à distance.

Ce modèle préfigure des architectures plus larges de diagnostic connecté, dans lesquelles le véhicule devient une source de données pour des services logiciels. Il met toutefois en évidence une dépendance essentielle : la qualité du traitement distant dépend de la fiabilité de l'acquisition initiale. Si les trames ou les paramètres extraits sont incomplets, mal horodatés ou insuffisamment structurés, la couche d'analyse ne peut pas produire un diagnostic fiable. Cette observation renforce la nécessité de concevoir une interface d'acquisition robuste avant de développer des fonctions avancées de traitement.

1.4.3 Tableaux de bord embarqués fondés sur CAN

La visualisation locale des données véhicule constitue une autre orientation de recherche. Pawlenka et al. présentent la conception d'une unité de tableau de bord numérique basée sur le bus CAN [22]. Dans ce type d'application, les trames ou paramètres issus du réseau embarqué sont transformés en informations affichables par l'utilisateur. L'objectif principal n'est pas uniquement l'acquisition, mais la restitution d'informations de conduite ou de diagnostic sous une forme exploitable.

Cette famille de travaux est pertinente pour les véhicules ne disposant pas d'un tableau de bord intelligent ou d'une interface de diagnostic moderne. Dans le cadre du présent projet, elle constitue un contexte d'intégration : le dispositif peut fournir un flux de données à une interface locale, mais la contribution centrale du mémoire demeure la couche d'acquisition elle-même.

1.4.4 Sécurité automobile et rétro-ingénierie CAN

La recherche en sécurité automobile a également contribué à montrer l'importance de l'accès aux réseaux internes du véhicule. Les travaux de Koscher et al. ont mis en évidence la possibilité d'analyser expérimentalement la sécurité d'un véhicule moderne à partir de ses réseaux embarqués [15]. Hoppe et al. ont étudié les menaces pesant sur les réseaux CAN automobiles, en soulignant que la compréhension du trafic interne constitue un préalable à l'analyse de sécurité [8]. Ces travaux ne relèvent pas directement du diagnostic de maintenance, mais ils démontrent que l'observation et l'interprétation des communications CAN sont des activités techniques importantes dans l'étude des véhicules modernes.

Miller et Valasek ont également montré, à travers une analyse des surfaces d'attaque automobiles, que les véhicules modernes exposent plusieurs interfaces susceptibles de conduire à des interactions avec les systèmes embarqués [19]. Cho et Shin ont proposé une méthode d'identification des calculateurs à partir de caractéristiques observables sur le réseau, dans une perspective de détection d'intrusion [2]. Ces recherches confirment que l'acquisition de trames CAN ne constitue pas seulement une opération de journalisation, mais un outil d'analyse permettant d'étudier le comportement des calculateurs, les échanges inter-systèmes et les limites des architectures embarquées.

Pour le présent mémoire, l'intérêt de cette littérature est double. D'une part, elle légitime les approches fondées sur l'observation du bus et l'analyse des trames comme des méthodes d'ingénierie reconnues. D'autre part, elle rappelle que l'accès au bus doit être maîtrisé avec rigueur, car la capacité d'émettre ou de modifier des messages sur un réseau véhicule peut avoir des conséquences fonctionnelles importantes. Le

dispositif proposé doit donc être pensé comme un outil d'acquisition et de diagnostic contrôlé, non comme un moyen d'intervention arbitraire sur les calculateurs.

Tableau 1.2 – Comparaison des travaux académiques liés à l'acquisition CAN et au diagnostic embarqué.

Référence	**Objectif principal**	**Approche utilisée**	**Limite par rapport au présent travail**
Wang et al. [28]	Acquisition de données véhicule et diagnostic de défauts	Système d'acquisition appliqué au bus véhicule	Moins centré sur la réalisation d'un dispositif OBD-II ouvert et reproductible
Chen et Li [1]	Acquisition et surveillance de données CAN	Environnement LabVIEW et supervision sur ordinateur	Approche orientée laboratoire, moins adaptée à un dispositif embarqué autonome
Jhou et al. [14]	Diagnostic OBD-II connecté	Transmission des données vers une infrastructure distante	Accent mis sur le traitement connecté plutôt que sur la couche matérielle d'acquisition
Pawlenka et al. [22]	Tableau de bord numérique fondé sur CAN	Visualisation embarquée des données véhicule	Objectif principal orienté affichage, non acquisition générique OBD-II/CAN

1.5 Limites identifiées dans la littérature et les solutions existantes

L'analyse des solutions existantes met en évidence plusieurs limites. La première concerne la différence entre accès fonctionnel et accès aux données brutes. Une valise de diagnostic peut offrir une couverture fonctionnelle importante, mais elle ne donne pas nécessairement accès aux trames échangées, aux mécanismes de transport ou aux détails de l'interaction entre l'outil et les calculateurs. À l'inverse, un dispositif d'acquisition CAN donne accès à la couche message, mais il doit ensuite résoudre le problème de l'interprétation des données.

La deuxième limite concerne les adaptateurs OBD-II grand public. Leur interface de haut niveau simplifie l'utilisation, mais réduit la maîtrise des échanges. Pour un usage de diagnostic standard, cette abstraction est acceptable ; pour un usage d'ingénierie, elle limite la capacité à observer les trames, à reconstruire les messages multi-trames et à contrôler précisément le comportement de l'outil. La documentation de l'ELM327 montre que ce composant agit comme un interpréteur entre les commandes de l'hôte et les protocoles OBD du véhicule [5]. Cette architecture explique son intérêt pour les applications simples, mais aussi ses limites pour une acquisition bas niveau.

La troisième limite concerne l'interprétation des trames CAN. Une trame brute contient un identifiant, une longueur et une charge utile, mais elle ne donne pas direc-

tement une grandeur physique exploitable. Le passage de l'octet brut à l'information interprétable nécessite des règles de décodage décrivant la position des signaux, leur longueur, leur facteur d'échelle, leur offset et leur unité [3]. Dans le cas des véhicules particuliers, une partie importante de ces règles est spécifique au constructeur. Les services standardisés OBD-II et UDS apportent donc une structure utile, mais ils ne suppriment pas totalement la difficulté d'accès aux données propriétaires.

La quatrième limite concerne la diversité des véhicules. Tous les véhicules équipés d'un connecteur de diagnostic ne reposent pas nécessairement sur la même couche physique ou le même protocole. Les véhicules récents utilisent largement le diagnostic sur CAN, tandis que certains véhicules plus anciens peuvent encore exploiter KWP2000 sur ligne K [11]. Cette diversité impose de concevoir un dispositif évolutif, capable de prendre en compte plusieurs scénarios d'acquisition ou, à défaut, de définir clairement son périmètre de compatibilité.

Enfin, les plateformes commerciales d'acquisition CAN montrent qu'un dispositif robuste ne se limite pas à recevoir des trames. Il doit intégrer des fonctions telles que l'horodatage, le filtrage, le stockage ou la transmission structurée des données [4]. Ces fonctionnalités constituent un repère utile pour définir les exigences d'une architecture académique, mais le présent travail se distingue par son objectif : construire un dispositif maîtrisé, reproductible et adapté au diagnostic via OBD-II, plutôt que d'utiliser directement un produit commercial prêt à l'emploi.

La différence principale réside ainsi dans le passage d'un produit fini, difficilement modifiable, à un dispositif d'acquisition reproductible dont les choix matériels et logiciels restent contrôlables dans un cadre académique.

Tableau 1.3 – Synthèse des limites identifiées et des exigences associées au dispositif proposé.

Limite identifiée	**Exigence associée pour le dispositif proposé**
Architecture fermée des outils professionnels	Concevoir une architecture ouverte et documentée, permettant de maîtriser la chaîne d'acquisition.
Accès limité aux trames brutes	Fournir un accès direct aux trames CAN, avec identifiant, DLC, charge utile et horodatage.
Interface de haut niveau des adaptateurs grand public	Réduire la dépendance aux interpréteurs OBD en utilisant une couche CAN plus directement contrôlable.
Gestion complexe des échanges multi-trames	Prévoir une gestion logicielle du transport ISO-TP pour les réponses dépassant la capacité d'une trame CAN classique.
Données constructeur difficiles à interpréter	Structurer les données acquises afin de permettre un traitement externe et une évolution vers des mécanismes de décodage plus avancés.
Hétérogénéité des véhicules et des protocoles	Définir une architecture évolutive et un périmètre de compatibilité clairement documenté.

1.6 Positionnement et contribution de ce travail

Le présent travail se positionne à l'intersection de trois familles de solutions : les outils de diagnostic, les enregistreurs CAN et les plateformes embarquées de traitement. Les outils de diagnostic fournissent une interface orientée utilisateur, mais masquent souvent les détails de la communication. Les enregistreurs CAN donnent accès aux trames, mais ne sont pas toujours conçus autour des services OBD-II et des besoins de diagnostic. Les plateformes académiques démontrent la faisabilité d'architectures d'acquisition et de supervision, mais elles sont souvent liées à un environnement logiciel ou matériel particulier.

La contribution recherchée est donc la conception d'un dispositif embarqué d'acquisition capable de se connecter au port OBD-II, d'interagir avec le bus CAN, de structurer les trames acquises et de fournir un flux exploitable par une couche de traitement externe. L'intérêt du dispositif réside dans son caractère ouvert, adaptable et évolutif. Il doit permettre de maîtriser les échanges de communication, de comprendre les limites des protocoles utilisés et de préparer l'intégration avec des fonctions de diagnostic plus avancées.

Le choix d'une architecture fondée sur un contrôleur CAN dédié s'inscrit dans cette logique. Il permet de rapprocher le système de la couche bus et de réduire la dépendance à une interface interprétée de haut niveau. Cette orientation facilite la capture passive, le contrôle de l'émission des requêtes, l'horodatage des trames et la mise en œuvre logicielle de mécanismes comme le transport ISO-TP. Elle fournit également une base plus flexible pour l'évolution future du système vers des services de diagnostic avancés.

Dans l'architecture globale du projet, le dispositif d'acquisition constitue la couche véhicule. Il peut alimenter une interface locale de visualisation embarquée, par exemple pour afficher des informations de diagnostic dans un véhicule ne disposant pas d'un tableau de bord intelligent. Il peut également transmettre les données vers une couche externe chargée de l'interprétation des codes défauts, de l'analyse et de la génération d'instructions. Ces fonctions de traitement constituent toutefois une couche distincte : le présent chapitre positionne principalement le dispositif comme une interface fiable entre le véhicule et les systèmes logiciels qui exploiteront ses données.

Ainsi, le travail ne vise pas à remplacer intégralement les outils professionnels existants, ni à construire une plateforme logicielle complète de diagnostic. Il vise à réaliser la couche fondamentale sans laquelle ces traitements ne peuvent pas être fiables : une acquisition maîtrisée des trames CAN et des échanges OBD-II, dans une architecture ouverte, reproductible et adaptée au contexte académique.

1.7 Conclusion

Ce chapitre a présenté l'évolution du diagnostic automobile, les principales catégories d'outils existants et les travaux académiques liés à l'acquisition CAN, au diagnostic embarqué et à l'analyse des réseaux véhicules. Il a montré que les solutions actuelles répondent chacune à un besoin spécifique, mais qu'elles ne fournissent pas toujours une couche d'acquisition ouverte, maîtrisée et adaptée à la construction d'une plateforme de diagnostic évolutive.

Le positionnement retenu pour ce mémoire consiste donc à concevoir un dispositif embarqué capable d'assurer l'acquisition fiable des trames CAN via l'interface OBD-II, tout en préparant l'intégration avec des couches de visualisation et d'analyse. Le chapitre suivant présente les fondements théoriques nécessaires à cette conception, notamment l'architecture électronique des véhicules, le bus CAN, l'interface OBD-II, le transport ISO-TP, les services UDS et les spécificités des protocoles utilisés sur les véhicules plus anciens.

CHAPITRE 2

Fondements théoriques et technologies du système VEYA

2.1 Introduction

Ce chapitre présente les fondements théoriques et technologiques nécessaires à la compréhension d'un système d'acquisition et d'analyse de données de diagnostic automobile. Il se limite aux principes généraux des architectures électroniques embarquées, des interfaces de diagnostic et des protocoles de communication associés.

L'objectif est de définir un cadre technique commun avant l'étude de conception. Les notions abordées concernent principalement les réseaux embarqués, l'accès OBD-II, le bus CAN et le transport ISO-TP, sans décrire l'architecture finale du système.

2.2 Architecture électronique des véhicules modernes

Les véhicules modernes reposent sur une architecture distribuée composée de plusieurs unités de contrôle électronique, généralement appelées UCE ou ECU. Ces calculateurs assurent des fonctions spécialisées liées au moteur, au freinage, à la sécurité, au confort ou à l'infodivertissement, et coopèrent à travers des réseaux embarqués [20, 23].

Cette organisation nécessite des mécanismes de communication fiables entre calculateurs. Les réseaux embarqués permettent d'échanger des informations de commande, d'état et de diagnostic, tandis que les passerelles assurent l'interconnexion entre domaines fonctionnels lorsque plusieurs bus coexistent dans le véhicule.

L'accès au diagnostic s'inscrit dans cette architecture comme un point d'observation et d'interrogation normalisé. Il ne donne pas nécessairement accès à l'ensemble des communications internes, mais fournit une interface structurée pour consulter certaines données et certains états de fonctionnement.

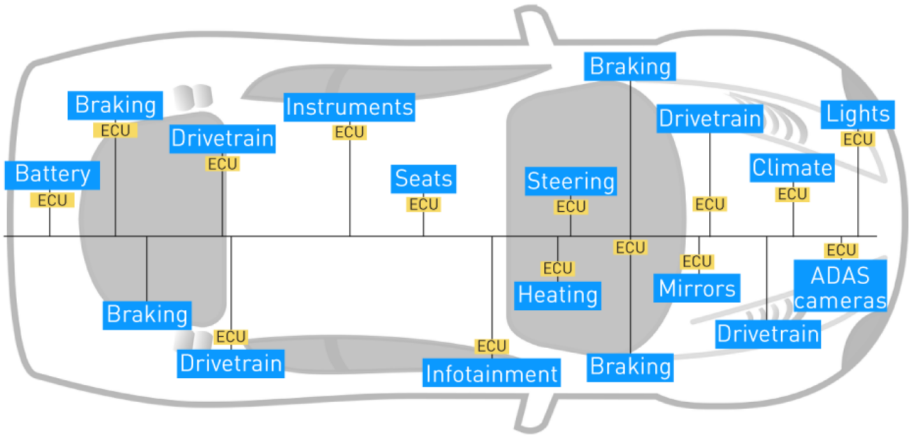


FIGURE 2.1 – Architecture électronique simplifiée d'un véhicule moderne intégrant plusieurs UCE, réseaux embarqués, passerelle et accès diagnostic.

2.3 Interface OBD-II et accès au diagnostic

L'interface OBD-II constitue le point d'accès standardisé au diagnostic embarqué sur les véhicules compatibles. Elle définit notamment un connecteur physique, des lignes de communication et un ensemble de services liés à la surveillance des systèmes antipollution et au diagnostic générique [25, 26, 10].

Le connecteur OBD-II regroupe plusieurs broches dont certaines sont réservées à l'alimentation, aux masses et aux protocoles de communication. Selon le véhicule et son année de production, les lignes disponibles peuvent inclure CAN, K-Line, L-Line ou les bus SAE J1850.

L'accès générique par OBD-II reste limité par rapport aux capacités internes des calculateurs. Il fournit un cadre commun pour certaines informations normalisées, mais les données constructeur, les fonctions étendues et les services spécifiques peuvent dépendre d'autres protocoles ou de bases techniques propres au véhicule.

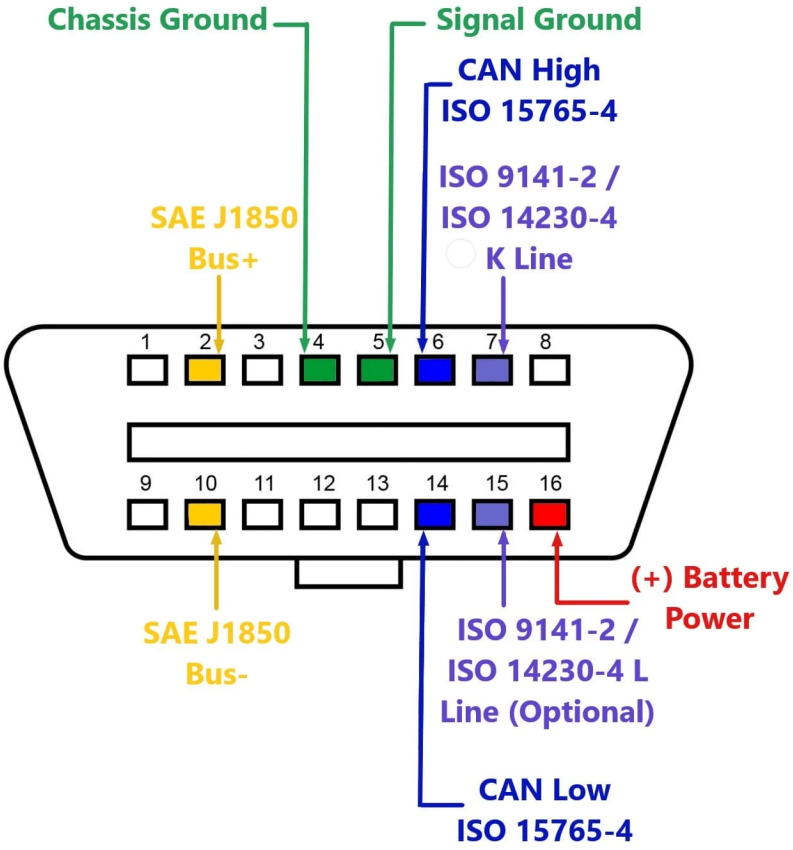


FIGURE 2.2 – Brochage du connecteur OBD-II et signaux utilisés pour l'accès au diagnostic.

Broche	Fonction
2	SAE J1850 Bus+
4	Masse châssis
5	Masse signal
6	CAN-H
7	K-Line
10	SAE J1850 Bus-
14	CAN-L
15	L-Line
16	Alimentation batterie +12 V

TABLEAU 2.1 – Broches OBD-II pertinentes pour le dispositif d'acquisition.

2.4 Bus CAN : principes, trames et couche physique

Le bus CAN est un réseau de communication série conçu pour les environnements embarqués exigeants. Il permet à plusieurs calculateurs de partager un même support de communication tout en conservant un mécanisme d'arbitrage déterministe fondé sur la priorité des identifiants [24, 9, 21].

La couche physique du CAN repose sur une transmission différentielle utilisant les lignes CAN-H et CAN-L. Cette approche améliore la robustesse face aux perturbations électromagnétiques, ce qui explique son adoption large dans les architectures automobiles [27, 3].

Une trame CAN classique comprend plusieurs champs, dont le début de trame, l'identifiant, les bits de contrôle, la longueur de données, les octets de données, le contrôle CRC, l'acquittement et la fin de trame. Il convient également de distinguer le contrôleur CAN, chargé du protocole logique, du transcepteur CAN, chargé de l'adaptation électrique au bus.

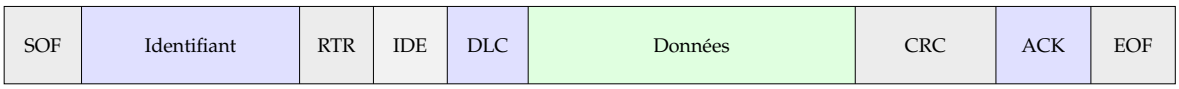


FIGURE 2.3 – Structure simplifiée d'une trame CAN classique.

Champ	**Rôle**
SOF	Indique le début de la trame et synchronise les nœuds.
Identifiant	Identifie le message et détermine sa priorité d'arbitrage.
RTR	Distingue une trame de données d'une trame de requête distante.
IDE	Indique si l'identifiant est standard ou étendu.
DLC	Indique le nombre d'octets transportés dans le champ de données.
Données	Contient la charge utile transmise par la trame.
CRC	Permet la détection d'erreurs de transmission.
ACK	Permet aux récepteurs de confirmer la réception correcte.
EOF	Marque la fin de la trame.

TABLEAU 2.2 – Rôle des principaux champs d'une trame CAN classique.



FIGURE 2.4 – Chaîne fonctionnelle entre l'unité de traitement, le contrôleur CAN, le transcepteur et le bus véhicule.

2.5 Transport ISO-TP sur CAN

La taille utile d'une trame CAN classique étant limitée, les messages de diagnostic dépassant cette capacité nécessitent une couche de transport. ISO-TP, défini par ISO 15765-2, fournit ce mécanisme de segmentation et de réassemblage pour transporter des messages plus longs sur CAN [12].

Le principe consiste à distinguer les messages courts, transmis dans une trame unique, des messages longs, transmis sous forme de première trame puis de trames consécutives. Le récepteur peut contrôler le rythme d'émission au moyen d'une trame de contrôle de flux [3].

Cette couche est essentielle pour les échanges de diagnostic modernes, car de nombreux services produisent des réponses dont la taille dépasse les quelques octets

disponibles dans une seule trame CAN. ISO-TP constitue ainsi un lien fonctionnel entre le bus CAN et les protocoles de diagnostic de plus haut niveau.

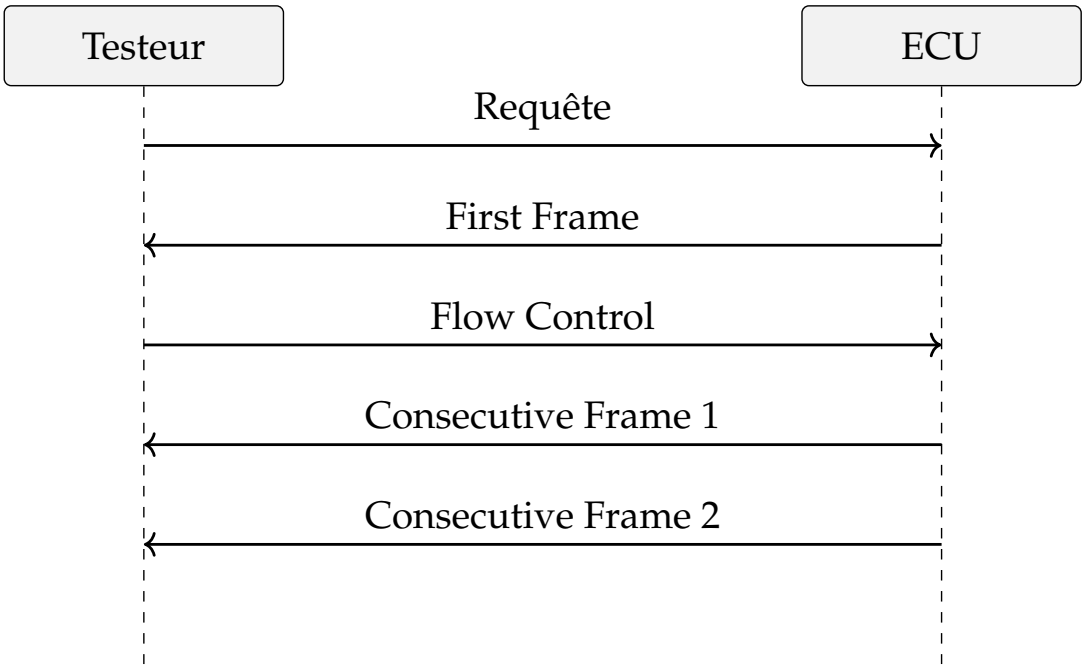


FIGURE 2.5 – Principe d'un échange ISO-TP multi-trame sur CAN.

Type de trame	**Identifiant PCI**	**Rôle**
Single Frame	0x0	Transporter un message court dans une seule trame CAN.
First Frame	0x1	Débuter un message plus long et annoncer sa longueur totale.
Flow Control	0x3	Réguler l'envoi des trames consécutives par le récepteur.
Consecutive Frame	0x2	Transporter les fragments successifs du message long.

TABLEAU 2.3 – Types de trames définis par ISO-TP.

2.6 Services UDS pour le diagnostic

Le protocole UDS, défini par l'ISO 14229, fournit un cadre normalisé pour les échanges de diagnostic entre un testeur et les calculateurs d'un véhicule [13]. Il organise la communication autour de services identifiés par un SID, chacun correspondant à une fonction de diagnostic telle que la lecture de données, la gestion de session ou l'accès aux informations de défaut.

UDS ne constitue pas un bus physique, mais une couche applicative de diagnostic. Il peut être transporté sur différents supports, notamment sur CAN au moyen d'ISO-TP, ce qui permet de séparer les règles de diagnostic des mécanismes de segmentation et de transport des messages.

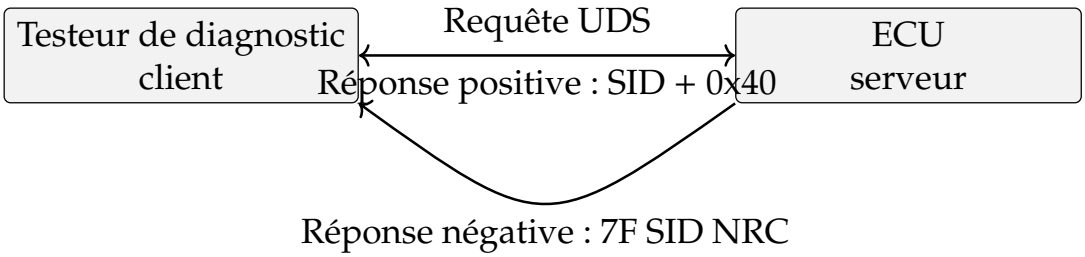


FIGURE 2.6 – Modèle client-serveur des services UDS.

SID	Service UDS	Rôle général
0x10	DiagnosticSessionControl	Sélectionner une session de diagnostic adaptée au type d'échange.
0x11	ECUReset	Demander une réinitialisation contrôlée d'un calculateur.
0x14	ClearDiagnosticInformation	Effacer des informations de diagnostic mémorisées.
0x19	ReadDTCInformation	Lire les informations relatives aux codes défaut.
0x22	ReadDataByIdentifier	Lire une donnée identifiée par un identifiant normalisé ou constructeur.
0x23	ReadMemoryByAddress	Lire une zone mémoire à partir d'une adresse.
0x24	ReadScalingDataByIdentifier	Lire les informations de mise à l'échelle associées à une donnée.
0x27	SecurityAccess	Gérer un accès sécurisé à certaines fonctions protégées.
0x28	CommunicationControl	Contrôler certains modes de communication du calculateur.
0x2A	ReadDataByPeriodicIdentifier	Lire des données transmises périodiquement par identifiant.
0x2C	DynamicallyDefineDataIdentifier	Définir dynamiquement des identifiants de données.
0x2E	WriteDataByIdentifier	Écrire une donnée identifiée dans un calculateur.
0x2F	InputOutputControlByIdentifier	Commander temporairement certaines entrées ou sorties.
0x31	RoutineControl	Lancer, arrêter ou interroger une routine interne.
0x34	RequestDownload	Demander l'ouverture d'un transfert vers le calculateur.
0x35	RequestUpload	Demander l'ouverture d'un transfert depuis le calculateur.
0x36	TransferData	Transférer des blocs de données pendant une opération de téléchargement ou téléversement.
0x37	RequestTransferExit	Terminer une séquence de transfert de données.
0x38	RequestFileTransfer	Demander un transfert de fichier.
0x3D	WriteMemoryByAddress	Écrire une zone mémoire à partir d'une adresse.
0x3E	TesterPresent	Maintenir une session de diagnostic active.
0x85	ControlDTCSetting	Activer ou désactiver certaines conditions d'enregistrement des codes défaut.

TABLEAU 2.4 – Principaux services UDS normalisés par l'ISO 14229.

Ce tableau présente les principaux services UDS à un niveau théorique. Dans le périmètre d'étude de VEYA v0.1, l'usage reste orienté vers les échanges de lecture et de diagnostic. Les services impliquant l'effacement, l'écriture, l'accès sécurisé, le transfert mémoire ou le contrôle de routines doivent être considérés comme sensibles et nécessitent un encadrement strict.

2.6.1 Principe général des services UDS

UDS repose sur un modèle client-serveur. Le testeur de diagnostic envoie une requête à un calculateur, puis le calculateur répond par une réponse positive lorsque le service est accepté ou par une réponse négative lorsque la demande est refusée ou impossible à traiter [13].

Chaque service est identifié par un SID. Dans le cas général, une réponse positive reprend le SID de la requête augmenté de 0x40, tandis qu'une réponse négative utilise le format 0x7F, suivi du SID concerné et d'un code de réponse négative. Cette

structure facilite l'interprétation des échanges et la distinction entre acceptation, rejet temporaire et erreur de condition.

Les sessions de diagnostic définissent le contexte fonctionnel dans lequel les services sont exécutés. Certains services peuvent être accessibles dans une session par défaut, alors que d'autres exigent une session étendue ou des conditions particulières. Cette notion permet de limiter l'accès aux fonctions sensibles et d'organiser progressivement les niveaux de diagnostic.

2.6.2 Services UDS utiles au diagnostic

Parmi les services utiles à une approche de diagnostic orientée lecture, DiagnosticSessionControl permet de sélectionner le contexte d'échange, ReadDataByIdentifier permet de demander des données identifiées, ReadDTCInformation permet de consulter les informations relatives aux défauts, et TesterPresent permet de maintenir une session active pendant une séquence de communication [13].

Ces services forment une base cohérente pour l'acquisition d'informations sans action corrective directe sur le calculateur. Ils doivent toutefois être employés dans le respect des conditions prévues par le protocole, car une réponse dépend du véhicule, du calculateur, de la session active et des droits associés.

À l'inverse, les services d'écriture, d'effacement, de contrôle de routine, de transfert mémoire ou d'accès sécurisé appartiennent à un périmètre plus sensible. Ils sont importants dans le cadre général du diagnostic UDS, mais leur usage nécessite une maîtrise stricte des effets possibles sur le calculateur.

2.6.3 Notion de code défaut

Un code défaut, ou DTC, est une information mémorisée par un calculateur lorsqu'une condition de panne ou d'anomalie est détectée. Il associe généralement une identification du défaut à un état de mémorisation et à un contexte de diagnostic, selon les règles prévues par le calculateur et par les normes applicables.

Dans cette thèse, la lecture des DTC est considérée comme une opération d'acquisition et de communication. L'objectif théorique de ce chapitre est donc de situer le rôle des codes défaut dans l'échange de diagnostic, sans développer ici les méthodes d'interprétation détaillée, les priorités de réparation ou les recommandations techniques qui relèvent de la couche d'analyse.

2.7 KWP2000 et communication K-Line

KWP2000, normalisé dans la famille ISO 14230, correspond à un protocole de diagnostic largement rencontré sur des véhicules plus anciens, notamment lorsque la communication s'effectue par K-Line [11]. Contrairement au CAN, la K-Line repose sur une liaison série monofilaire, avec des contraintes de débit, d'initialisation et de temporisation plus marquées.

Ce contexte reste important pour un chapitre de fondements, car le parc automobile peut contenir des véhicules qui ne s'appuient pas exclusivement sur le diagnostic CAN moderne. L'étude des architectures de diagnostic doit donc tenir compte de

protocoles hérités, même lorsque le CAN et l'UDS constituent aujourd'hui des références dominantes.

La communication K-Line impose une attention particulière aux phases d'initialisation, aux temps de réponse et à l'adaptation électrique de la ligne. Elle illustre aussi la diversité des couches de diagnostic pouvant coexister autour du connecteur OBD-II, dont le brochage prévoit notamment des lignes dédiées à K-Line et L-Line [25, 26, 10].

Certaines couches de diagnostic anciennes ou spécifiques à un constructeur peuvent employer des services tels que 0x17 ou 0x18 pour des informations relatives aux défauts, tandis que l'UDS moderne utilise le service 0x19, ReadDTCInformation.

Critère	**CAN / ISO 15765**	**K-Line / KWP2000**
Support physique	Bus différentiel CAN-H/CAN-L.	Ligne série K-Line monofilaire.
Vitesse typique	Débit plus élevé, couramment 250 kbit/s ou 500 kbit/s.	Débit plus faible, typiquement 10,4 kbit/s.
Structure réseau	Communication multi-noeuds sur bus partagé.	Communication point à point ou réseau plus limité.
Véhicules concernés	Véhicules récents et architectures OBD-II modernes.	Véhicules plus anciens ou calculateurs hérités.
Intérêt pour VEYA	Base principale pour l'acquisition diagnostic moderne.	Compatibilité potentielle avec des véhicules moins récents.

Tableau 2.5 – Comparaison synthétique entre diagnostic sur CAN et communication K-Line/KWP2000.

2.8 Architecture matérielle utilisée dans le projet

L'architecture matérielle étudiée s'appuie sur des interfaces capables d'accéder au connecteur OBD-II, de dialoguer avec les réseaux de diagnostic et de transmettre les informations vers une unité de traitement. Les composants présentés dans cette section sont décrits pour leur rôle technique général, sans détailler le flux complet du système.

Deux familles d'approches peuvent être distinguées. La première utilise des adaptateurs de diagnostic prêts à l'emploi, qui masquent une partie des couches basses. La seconde s'appuie sur une cible embarquée associée à des circuits spécialisés, afin de mieux comprendre la chaîne matérielle entre le connecteur, le contrôleur de bus et l'interface électrique.

Interface	Rôle dans le projet	Intérêt	Limite
ELM327 Bluetooth clone	Premières expérimentations OBD-II.	Mise en oeuvre rapide et coût réduit.	Fiabilité et transparence limitées.
ELM327 USB clone	Essais de communication filaire.	Connexion plus stable que Bluetooth.	Dépendance au firmware de l'adaptateur.
OBDLink SX	Interface de communication avancée.	Communication plus stable observée pendant les essais VEYA.	Accès encore abstrait par l'adaptateur.
ESP32 + MCP2515 + transcepteur CAN	Cible embarquée CAN envisagée.	Maîtrise plus directe de la chaîne matérielle.	Intégration matérielle et firmware à étudier.
ESP32 + L9637D K-Line	Cible K-Line envisagée.	Extension vers les véhicules plus anciens.	Protocole et temporisations plus contraignants.

TABLEAU 2.6 – Évolution des interfaces d'acquisition utilisées ou envisagées dans le projet.

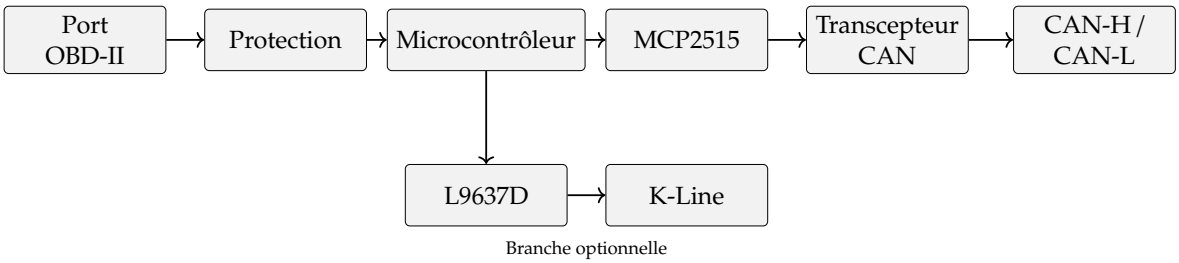


FIGURE 2.7 – Chaîne matérielle générique d'un dispositif d'acquisition CAN/K-Line.

2.8.1 Adaptateurs ELM327 et premières expérimentations

Les adaptateurs fondés sur l'ELM327 fournissent une interface de haut niveau entre un ordinateur ou un système embarqué et le connecteur OBD-II. Ils interprètent des commandes textuelles et les traduisent vers différents protocoles de diagnostic, ce qui facilite les premiers essais de communication avec un véhicule ou un simulateur [5].

Dans un contexte d'étude, les clones ELM327 Bluetooth et USB constituent des moyens accessibles pour observer des réponses OBD-II simples. Ils permettent de vérifier les principes généraux d'une interrogation de diagnostic, tout en évitant d'implémenter immédiatement les couches basses du bus.

Cette simplicité implique toutefois une limite importante. L'interpréteur et le firmware de l'adaptateur masquent une partie du comportement réel des trames, ce qui réduit le contrôle disponible sur les échanges bas niveau, les temporisations et certains scénarios multi-trames.

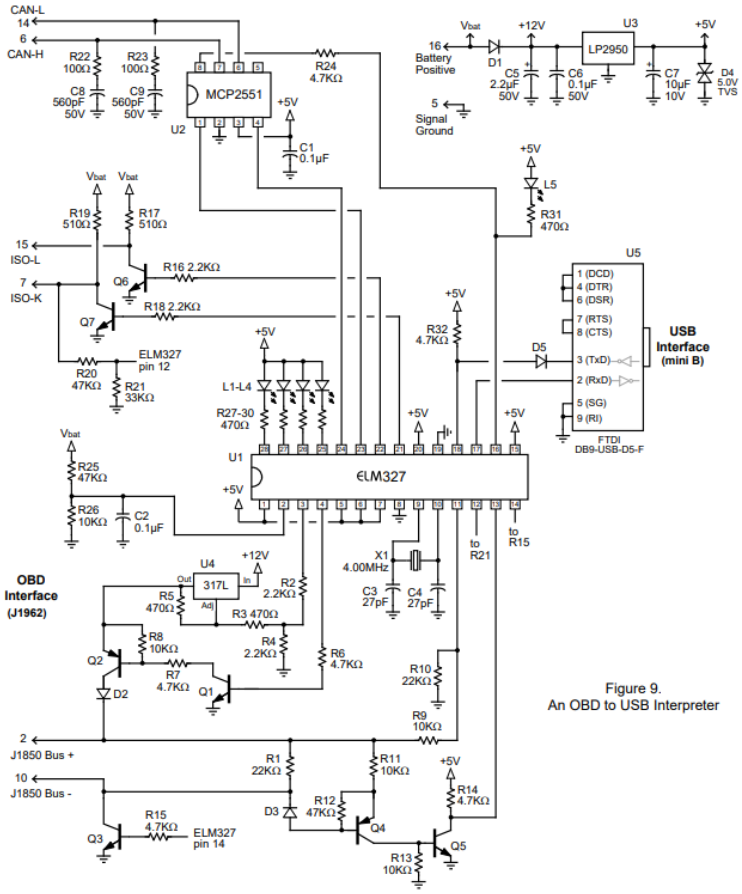


FIGURE 2.8 – Exemple d'architecture détaillée d'un adaptateur de diagnostic fondé sur l'ELM327.

2.8.2 OBDLink SX comme interface de communication avancée

L'OBDLink SX appartient à la même logique d'interface de communication OBD-II. Dans le cadre des essais VEYA, cette interface a offert une communication plus stable que les clones ELM327 Bluetooth utilisés initialement. Sa liaison filaire réduit certaines incertitudes rencontrées avec les interfaces génériques de faible qualité.

Comme pour les interfaces ELM327, ce type d'adaptateur reste néanmoins une couche d'abstraction. Il est adapté aux essais de communication et à l'observation de services de diagnostic courants, mais il ne remplace pas l'étude du contrôleur CAN, du transcepteur et du comportement électrique du bus.



FIGURE 2.9 – Adaptateur OBDLink SX utilisé comme interface de communication avancée.

2.8.3 Famille ESP32 comme cible embarquée

La famille ESP32 constitue une plateforme microcontrôleur adaptée aux prototypes d'étude nécessitant une capacité de traitement locale, des interfaces de communication et une intégration embarquée compacte. Dans une chaîne d'acquisition, elle peut assurer la coordination entre les interfaces physiques, le traitement initial des messages et la communication avec d'autres composants logiciels. Le choix précis de l'ESP32-S3 pour VEYA est présenté dans le chapitre de conception.

Son intérêt principal est de rapprocher l'acquisition du bus véhicule tout en conservant une architecture programmable. Le microcontrôleur ne réalise cependant pas seul l'adaptation CAN complète lorsque l'on utilise un contrôleur externe et un transcepteur distincts, ce qui impose de considérer séparément les fonctions logiques et électriques.

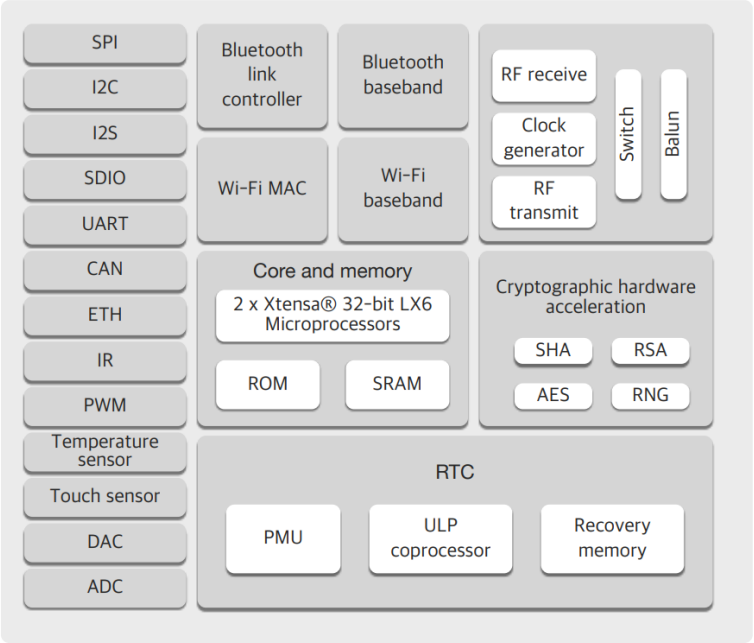


FIGURE 2.10 – Schéma fonctionnel simplifié de la famille ESP32 comme cible embarquée.

2.8.4 Contrôleur CAN MCP2515

Le MCP2515 est un contrôleur CAN externe qui prend en charge les mécanismes logiques du protocole CAN, notamment la construction des trames, les filtres d'identifiants, les buffers de transmission et de réception, ainsi que la gestion d'une partie des erreurs. Il communique avec le microcontrôleur par une interface SPI [18].

Dans une architecture modulaire, ce composant permet d'ajouter une interface CAN à une cible qui ne dispose pas directement du contrôleur requis ou lorsque l'on souhaite isoler clairement la fonction de contrôle du bus. Il ne se connecte toutefois pas directement aux lignes CAN-H et CAN-L, car cette adaptation relève du transcepteur.

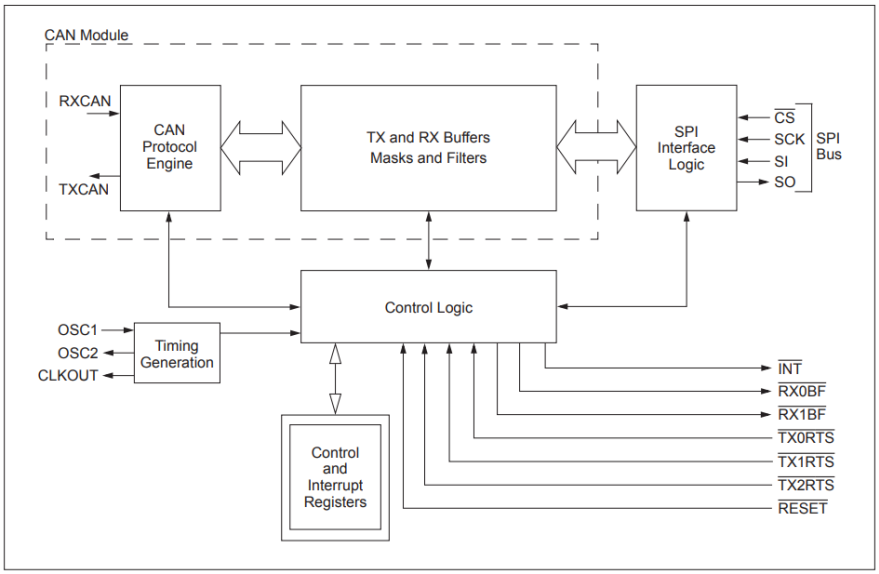


FIGURE 2.11 – Schéma fonctionnel du contrôleur CAN MCP2515.

2.8.5 Transcepteur CAN

Le transcepteur CAN assure l'interface électrique entre le contrôleur CAN et le bus physique. Il convertit les niveaux logiques du contrôleur en signal différentiel sur CAN-H et CAN-L, et réalise l'opération inverse lors de la réception [9, 27].

Cette distinction entre contrôleur et transcepteur est essentielle pour comprendre la couche physique. Le contrôleur traite les trames et l'arbitrage, tandis que le transcepteur supporte les contraintes électriques du bus, notamment les niveaux dominants et récessifs, la robustesse aux perturbations et l'interconnexion avec le réseau véhicule.

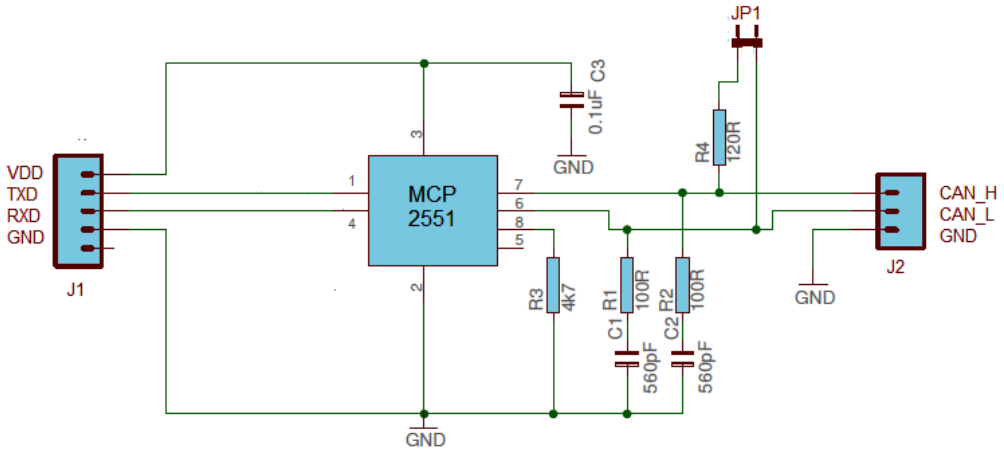


FIGURE 2.12 – Schéma fonctionnel d'un transcepteur CAN.

2.8.6 Interface K-Line

Une interface K-Line adapte les niveaux logiques d'un microcontrôleur à la ligne de diagnostic monofilaire utilisée par certains véhicules plus anciens. Elle joue un rôle comparable à celui d'un transcepteur, mais dans un contexte électrique et protocolaire différent du CAN [11].

L'intérêt de cette interface est de conserver une ouverture vers des calculateurs hérités qui ne communiquent pas nécessairement par CAN pour le diagnostic. Les contraintes de temporisation, d'initialisation et de débit rendent cependant cette communication plus spécifique que les échanges UDS sur CAN.

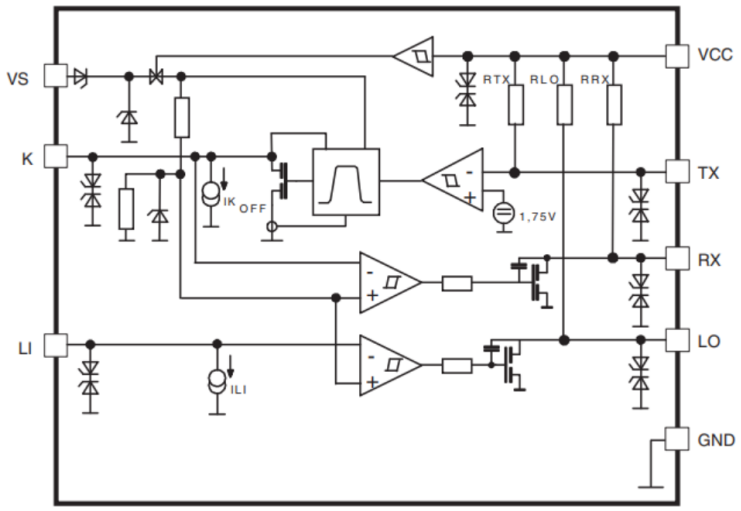


FIGURE 2.13 – Schéma fonctionnel du circuit L9637D pour l'interface K-Line.

2.8.7 Raspberry Pi 5 et écran embarqué

Le Raspberry Pi 5 peut être considéré comme une plateforme d'intégration et d'affichage pour un environnement embarqué de diagnostic. Sa capacité de traitement, son système d'exploitation généraliste et ses interfaces d'entrée-sortie le rendent adapté à l'exécution de couches applicatives, de services d'observation et d'interfaces utilisateur.

L'écran embarqué fournit un contexte de visualisation locale des informations acquises ou préparées par les couches logicielles. Dans ce chapitre, il est présenté comme un support matériel d'intégration et non comme la contribution principale du travail.

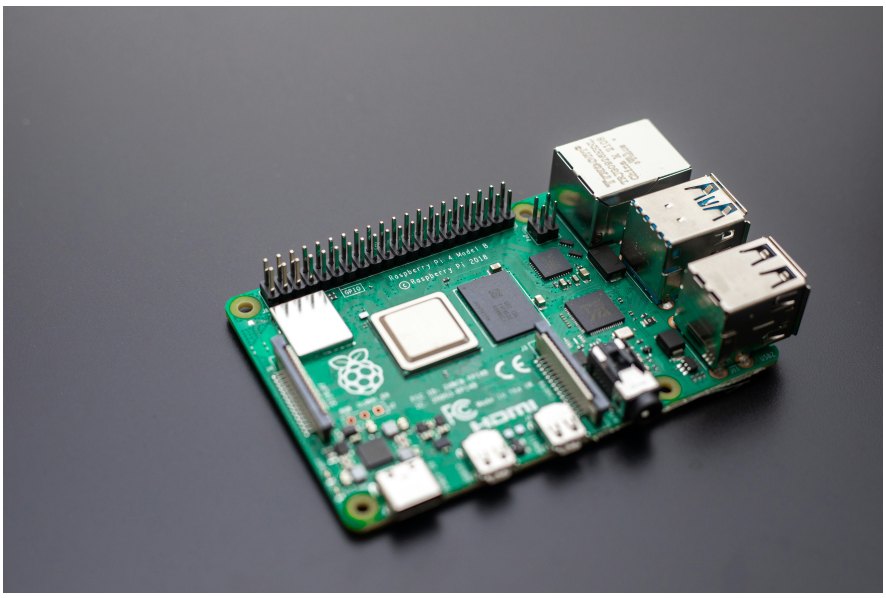


FIGURE 2.14 – Carte Raspberry Pi 5 envisagée pour l'interface embarquée.

2.9 Environnement logiciel et outils de développement

L'environnement logiciel regroupe les langages, bibliothèques et outils nécessaires à l'observation, à l'acquisition, au stockage et à la présentation des données de diagnostic. Cette section les présente comme des fondements techniques, sans détailler les choix d'implémentation propres aux chapitres suivants.

La diversité des outils reflète la nature transversale du diagnostic embarqué. Les scripts facilitent les essais de communication, les bases de données structurent les informations techniques, les outils de capture rendent les échanges observables, et les technologies d'interface permettent de présenter les données à l'utilisateur.

Outil ou technologie	**Rôle dans le projet**	**Niveau d'utilisation**
Python	Scripts d'essai, automatisation et prototypage.	Principal
PySerial	Communication série avec les interfaces de diagnostic.	Principal
SQLite	Stockage local de données structurées.	Principal
DDT2000	Source de données techniques pour l'analyse diagnostic.	Référence
Wireshark	Capture et analyse des échanges.	Analyse
USBPcap	Capture du trafic USB sous Windows.	Essai
Qt/QML	Développement de l'interface utilisateur.	Principal
C/C++	Développement applicatif et firmware embarqué.	Principal
CMake	Configuration et construction des composants logiciels.	Support
Bash	Automatisation de commandes et scripts d'environnement.	Support
Git/GitHub	Gestion de versions et collaboration.	Principal

Tableau 2.7 – Environnement logiciel et outils utilisés dans le projet.

2.9.1 Python et PySerial

Python est adapté aux scripts d'expérimentation, car il permet de décrire rapidement des séquences d'échange, de journaliser des réponses et de manipuler des données structurées. Dans le contexte du diagnostic, il peut être associé à des bibliothèques orientées CAN ou ISO-TP lorsque l'environnement matériel le permet [6, 16].

PySerial fournit une interface de communication série utile avec les adaptateurs exposés sous forme de port COM ou équivalent. Il permet d'envoyer des commandes, de lire des réponses et de construire des outils d'essai simples sans développer immédiatement une application complète.

2.9.2 SQLite et base DDT2000

SQLite est un moteur de base de données léger, intégré et adapté au stockage local d'informations structurées. Son usage convient aux environnements où les données

doivent être consultées rapidement sans déployer un serveur de base de données séparé.

La base DDT2000 représente une source technique associée à certains véhicules et calculateurs. Dans une approche de diagnostic, ce type de ressource peut servir à organiser des identifiants, des paramètres et des informations descriptives, tout en nécessitant une interprétation prudente selon le véhicule et le contexte.

2.9.3 Wireshark et USBPcap

Wireshark est un outil d'observation des échanges qui facilite l'analyse de paquets, de flux et de protocoles. Dans un contexte de diagnostic, il permet de comprendre la structure temporelle et logique de certaines communications, notamment lorsque les données transitent par une interface réseau ou par une capture compatible.

USBPcap complète cette approche sous Windows en permettant la capture de trafic USB. Il est utile lorsque l'interface de diagnostic communique avec l'ordinateur par USB et que l'on souhaite observer les échanges entre le logiciel hôte et l'adaptateur.

2.9.4 Qt/QML, C++ et JavaScript

Qt fournit un cadre de développement applicatif multiplateforme, tandis que QML facilite la description d'interfaces graphiques réactives. C++ peut assurer les traitements applicatifs, la gestion des modèles de données et l'intégration avec des bibliothèques système.

JavaScript intervient dans QML pour décrire certains comportements d'interface et des transformations simples de données. Ces technologies sont mentionnées ici comme base d'une interface applicative, sans préjuger de l'organisation interne détaillée de l'application.

2.9.5 C/C++ pour le firmware embarqué

Le C et le C++ restent courants dans le développement embarqué, car ils donnent un contrôle précis sur la mémoire, les interfaces matérielles, les interruptions et les temporisations. Ils sont donc adaptés aux composants qui doivent interagir avec un contrôleur CAN, une interface série ou une couche de communication bas niveau.

Dans un firmware de diagnostic, ces langages permettent de structurer les pilotes, les buffers et les états de communication. Leur usage exige cependant une attention particulière à la robustesse, aux erreurs de transmission et à la séparation entre logique applicative et accès matériel.

2.9.6 CMake, Bash, Git et GitHub

CMake sert à décrire la configuration de construction d'un projet C ou C++ de manière relativement portable. Bash permet d'automatiser des commandes répétitives, de préparer des environnements et d'enchaîner des opérations de compilation ou d'essai.

Git assure la gestion de versions du code et des documents, tandis que GitHub fournit un support de collaboration, de sauvegarde distante et de suivi des modifi-

cations. Ces outils ne relèvent pas directement du diagnostic automobile, mais ils structurent le travail d'ingénierie associé au projet.

2.10 Protocoles et formats de communication logiciels

Les données issues du diagnostic doivent être représentées et transportées entre plusieurs composants logiciels. Les formats et protocoles présentés dans cette section fournissent les bases nécessaires à des échanges lisibles, fiables ou interactifs, selon le besoin.

Il convient de distinguer le format de représentation, qui décrit la structure des données, du protocole de transport, qui organise leur circulation entre processus ou machines. JSON, TCP et WebSocket répondent à ces besoins à des niveaux différents.

Technologie	**Couche ou format**	**Rôle**	**Intérêt dans VEYA**
JSON	Format de représentation des données.	Structurer les messages et résultats.	Lisibilité et interopérabilité.
TCP	Transport réseau fiable.	Acheminer les flux entre composants.	Fiabilité des échanges.
WebSocket	Communication applicative bidirectionnelle.	Échanger des données en temps réel.	Interaction continue avec l'interface.

Tableau 2.8 – Comparaison des formats et protocoles logiciels utilisés pour l'échange des données.

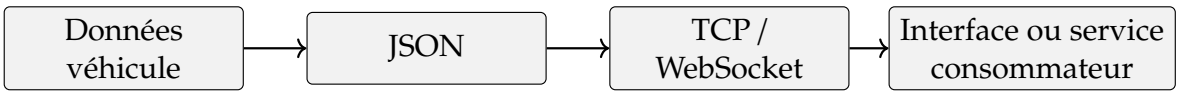


Figure 2.15 – Positionnement des formats et protocoles logiciels dans une chaîne d'échange de données.

2.10.1 JSON

JSON est un format textuel léger permettant de représenter des objets, des listes, des chaînes et des valeurs numériques. Sa lisibilité facilite l'inspection des messages et son usage est courant dans les interfaces logicielles modernes.

Dans un contexte de diagnostic, JSON peut servir à structurer des requêtes, des réponses, des paramètres ou des résultats d'acquisition. Il ne définit pas à lui seul la sémantique des données, mais fournit une représentation simple pour les échanger entre composants.

2.10.2 TCP

TCP est un protocole de transport orienté connexion qui fournit un flux fiable et ordonné entre deux extrémités. Il prend en charge la retransmission, le contrôle de flux et la détection de pertes, ce qui simplifie la conception des échanges applicatifs.

Pour une chaîne logicielle de diagnostic, TCP peut transporter des messages structurés entre un service d'acquisition, une application ou un outil d'observation. Il ne définit pas le format des messages, mais fournit le canal fiable sur lequel ceux-ci peuvent être transmis.

2.10.3 WebSocket

WebSocket établit une communication bidirectionnelle persistante au-dessus d'une connexion réseau. Il est adapté aux échanges applicatifs où des mises à jour doivent être transmises sans ouvrir une nouvelle requête pour chaque message.

Dans un environnement de supervision ou d'interface, ce mécanisme permet de diffuser des états, des mesures ou des événements vers un consommateur logiciel. Son intérêt réside dans l'interactivité, tandis que la cohérence des données dépend du format et des règles applicatives définis au-dessus du canal.

2.11 Conclusion

Ce chapitre a présenté les fondements nécessaires à la compréhension des architectures électroniques automobiles, des interfaces de diagnostic, des protocoles CAN, ISO-TP, UDS et KWP2000, ainsi que des principaux composants matériels et logiciels mobilisables dans un système d'acquisition. Ces éléments établissent le cadre technique à partir duquel peut être abordée la conception du système VEYA au chapitre suivant.

CHAPITRE 3

Conception du système embarqué VEYA

3.1 Introduction

Ce chapitre présente la conception des deux premiers niveaux du système VEYA : le dispositif d'acquisition connecté au port OBD et le tableau de bord local exécuté sur Raspberry Pi. Cette chaîne embarquée locale constitue le périmètre central du chapitre. La couche logicielle externe de diagnostic distant et de génération de rapports est déjà intégrée à VEYA, sans constituer la contribution embarquée centrale.

3.2 Besoins fonctionnels et objectifs du système

VEYA est conçu comme un système embarqué de diagnostic automobile multi-véhicules. Le périmètre principal de ce mémoire d'Ingéniorat porte sur deux niveaux complémentaires : le dispositif d'acquisition relié au véhicule et le tableau de bord exécuté sur Raspberry Pi 5. Une couche logicielle distante complète cette chaîne locale sans constituer la contribution embarquée centrale.

3.2.1 Fonctions principales

La première fonction attendue est l'acquisition en temps réel des données du véhicule à travers l'interface OBD-II et le bus CAN. Le dispositif d'acquisition ESP32-S3 constitue l'interface entre les calculateurs accessibles et le terminal embarqué. Il transmet les données au Raspberry Pi 5 par une liaison *Bluetooth Low Energy* (BLE), sans imposer de câblage supplémentaire entre le module placé au niveau du port OBD-II et le tableau de bord.

Le tableau de bord assure l'affichage en temps réel des paramètres de télémétrie : régime moteur, vitesse, température du liquide de refroidissement, position du papillon, charge moteur, tension de la batterie, niveau de carburant et température d'admission. Il fournit également une interface locale de diagnostic permettant la lecture des codes défaut (*Diagnostic Trouble Codes*, DTC), leur effacement et la lecture du kilométrage.

L'interface tactile regroupe les fonctions nécessaires à l'utilisation embarquée du terminal. Elle permet l'appairage Bluetooth, la configuration de l'accès au réseau Wi-Fi et la gestion d'un profil conducteur et véhicule. Le système doit aussi gérer automatiquement la source de données affichée : une source de démonstration est

utilisée en l'absence du dispositif réel, puis remplacée par les données issues de l'ESP32-S3 lorsque celui-ci est connecté.

3.2.2 Diagnostic distant connecté

La chaîne locale est complétée par un service distant de diagnostic déjà intégré au système. Ce service s'appuie sur un serveur Django et une base PostgreSQL pour produire un rapport structuré à partir des codes défaut. Le rapport présente la signification des codes, leurs causes probables, les symptômes associés et les réparations recommandées.

Une session de diagnostic à distance permet également à un expert d'accéder au système depuis un navigateur. L'interface distante lui permet d'observer la télémétrie en temps réel, de demander une lecture des DTC et de consulter le détail des codes. Cette capacité reste une couche logicielle externe connectée à la plateforme embarquée. Sa validation de bout en bout sera présentée dans le chapitre 4.

3.2.3 Objectifs de conception

La modularité constitue un objectif structurant de VEYA. L'architecture sépare la couche d'acquisition, la couche de transport, le cœur de traitement, la couche de présentation et le service externe de diagnostic. Chaque niveau communique au moyen d'interfaces définies afin de limiter les dépendances et de faciliter les évolutions du système.

La robustesse guide également la conception du cœur logiciel. Une machine d'états prend en charge la sélection de la source de données. Lorsque le dispositif réel se déconnecte, le système revient automatiquement à la source de démonstration. Ce mécanisme permet de maintenir le fonctionnement de l'interface locale et de gérer les transitions sans intervention de l'utilisateur.

L'architecture vise enfin un usage multi-véhicules. Elle doit permettre d'adapter la chaîne d'acquisition à différents contextes véhicule et protocolaires sans remettre en cause la séparation des couches. Cette exigence s'accompagne d'une discipline de preuve : aucune correspondance entre un code défaut, un bit ou une interprétation fonctionnelle ne doit être considérée comme correcte sans validation empirique. Les méthodes expérimentales appliquées à cette vérification seront exposées dans le chapitre 4.

3.3 Vue générale de l'architecture VEYA

L'architecture globale de VEYA est organisée autour d'une chaîne locale embarquée, complétée par une couche logicielle distante. Le périmètre central de ce mémoire d'Ingéniorat couvre le dispositif d'acquisition relié au véhicule et le terminal Raspberry Pi 5 assurant la restitution locale des données. La couche distante est intégrée au système, mais elle reste externe à cette chaîne embarquée.

Les calculateurs du véhicule constituent les sources d'information. Les échanges accessibles transitent par l'interface OBD-II et le bus CAN avant d'être pris en charge par le dispositif d'acquisition ESP32-S3. Ce dispositif correspond à l'architecture

d'acquisition personnalisée retenue pour le prototype. Sa conception sera présentée dans la section 3.5 sans anticiper les résultats expérimentaux du chapitre suivant.

La liaison *Bluetooth Low Energy* (BLE) relie le dispositif d'acquisition au Raspberry Pi 5. Sur ce terminal, le processus *bridge* relaie les données vers le *core*, puis le service *ws_server* les diffuse vers le tableau de bord Qt6/QML exécuté en mode kiosque. L'utilisateur dispose ainsi d'une interface locale dédiée à la visualisation et au diagnostic.

Une chaîne distante complète cette organisation. Depuis le Raspberry Pi, un relais WebSocket communique à travers Internet et un tunnel Cloudflare avec un serveur Django associé à une base PostgreSQL. Cette couche fournit le service distant de diagnostic, les rapports structurés et l'interface consultée par l'expert depuis un navigateur. Elle est intégrée et validée de bout en bout, tout en demeurant une couche logicielle externe au système embarqué local.

La figure 3.1 synthétise cette organisation. La chaîne distante est représentée en pointillés afin de distinguer son positionnement externe, et non pour indiquer une absence d'implémentation.

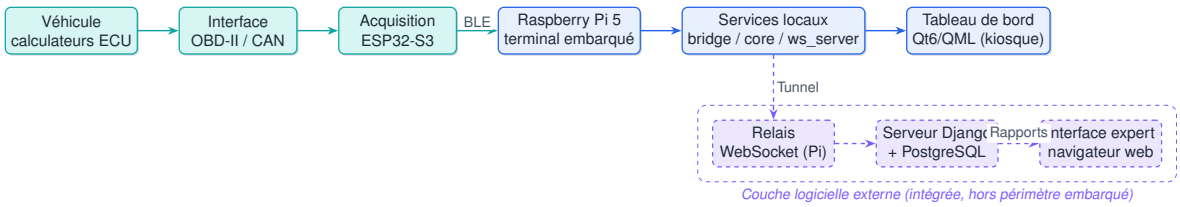


FIGURE 3.1 – Architecture globale de VEYA : chaîne embarquée locale et service distant de diagnostic.

Cette vue d'ensemble fixe les responsabilités de chaque niveau. Les choix relatifs au tableau de bord, au dispositif d'acquisition, à la liaison BLE et à la chaîne protocolaire sont présentés séparément dans les sections suivantes. Les détails de mise en oeuvre et de validation de la chaîne distante seront abordés dans le chapitre 4.

3.4 Conception du tableau de bord embarqué

3.4.1 Rôle du tableau de bord embarqué

Le tableau de bord constitue le terminal utilisateur embarqué de VEYA. Il est exécuté sur un Raspberry Pi 5 associé à un écran tactile. Son rôle est de rendre accessibles, dans l'habitacle, les informations acquises depuis le véhicule ainsi que les fonctions locales de diagnostic.

L'application fonctionne en mode kiosque sur Linux, avec un affichage direct sur le framebuffer *linuxfb*. Le démarrage automatique conduit directement à l'interface VEYA. L'utilisateur n'interagit donc pas avec l'environnement Linux : le système se comporte comme un terminal automobile dédié.

3.4.2 Organisation logicielle de l'interface

L'interface graphique repose sur Qt6 et QML. Cette couche de présentation reste distincte des services chargés de recevoir, sélectionner et diffuser les données. Cette

séparation limite les dépendances entre l'affichage et les moyens d'acquisition.

Sur le Raspberry Pi, le processus *bridge* assure le relais entre la liaison BLE et le *core*. Le *core* constitue le service central : il prend en charge la source active et la circulation des données vers l'interface. Le service *ws_server* assure leur diffusion locale par WebSocket. Dans la couche Qt6/QML, le fournisseur de données *VehicleDataProvider* centralise les informations reçues et les met à disposition des écrans de l'application.

Cette organisation permet de faire évoluer l'interface sans coupler directement les composants graphiques au dispositif OBD. Elle permet également d'intégrer les fonctions de rapport et de session distante tout en conservant la chaîne locale comme noyau du terminal embarqué.

3.4.3 Profils et écrans fonctionnels

L'interface est organisée autour de profils correspondant à des usages distincts. Le profil conducteur est destiné à la visualisation de la télémétrie en temps réel et des avertissements associés. Il présente les informations utiles à la conduite et à la surveillance locale du véhicule.

Le profil média réserve un espace fonctionnel pour une extension de l'interface. Il est identifié dans l'organisation du tableau de bord, mais son développement détaillé ne relève pas du périmètre de ce mémoire.

Le profil diagnostic regroupe les fonctions liées à l'état du véhicule : lecture des DTC, effacement des DTC, lecture du kilométrage, génération d'un rapport structuré et ouverture d'une session de diagnostic à distance. Cette session permet à un expert d'observer la télémétrie, de demander une lecture des DTC et de consulter leur détail depuis un navigateur.

La figure 3.2 présente les écrans principaux de l'interface.

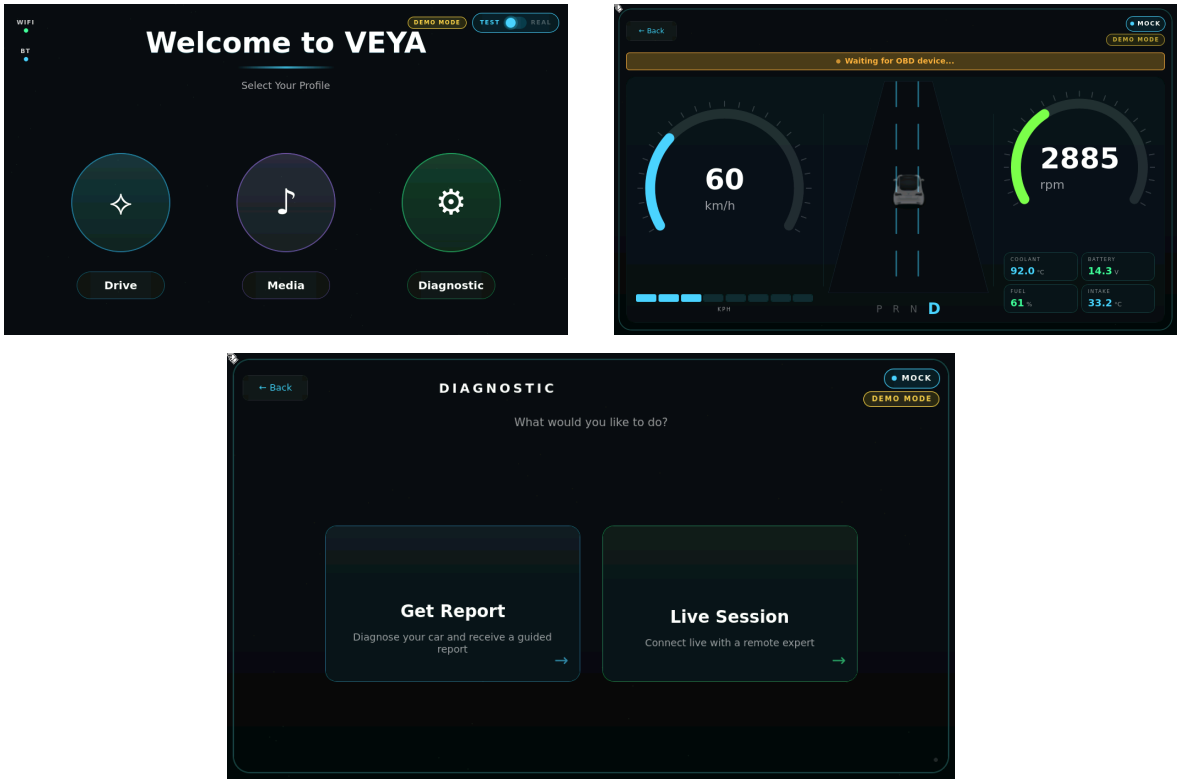


FIGURE 3.2 – Écrans principaux de l'interface VEYA : accueil, conduite et diagnostic.

Des écrans de configuration complètent ces profils. L'utilisateur peut réaliser l'appairage Bluetooth depuis l'écran tactile, configurer l'accès au réseau Wi-Fi et gérer un profil conducteur et véhicule.

La figure 3.3 illustre les écrans de configuration Wi-Fi et Bluetooth.

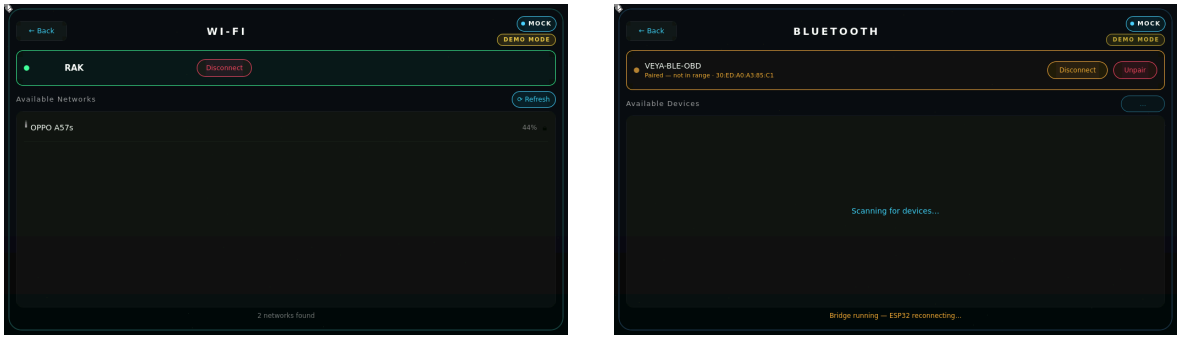


FIGURE 3.3 – Écrans de configuration de la connectivité du tableau de bord VEYA : Wi-Fi et Bluetooth.

3.4.4 Gestion des sources de données

Le tableau de bord doit rester exploitable lorsque le dispositif d'acquisition réel n'est pas disponible. Le *core* s'appuie pour cela sur une machine d'états qui distingue la source de démonstration et la source réelle transmise par l'ESP32-S3 à travers la liaison BLE.

Lorsque le dispositif réel est disponible, le système bascule vers les données acquises depuis le véhicule. En cas de déconnexion, il revient automatiquement à

la source de démonstration. Ce principe de repli permet de conserver une interface active sans imposer une intervention de l'utilisateur. Le basculement a été vérifié dans les deux sens : connexion du dispositif réel vers la source ESP32-S3, puis retour automatique vers la source de démonstration lors de la déconnexion.

3.4.5 Intégration mécanique et ergonomie

Le Raspberry Pi 5, l'écran tactile et leur boîtier forment un terminal destiné à une utilisation dans l'habitacle. La conception mécanique doit prendre en compte l'accès tactile à l'interface, la lisibilité de l'affichage et l'intégration physique du terminal.

Le boîtier du tableau de bord est conçu pour un écran tactile de 7 pouces en orientation paysage, avec une définition de 1024×600. Il est réalisé en PLA et doit maintenir l'accès direct à la surface tactile. L'écran est raccordé au Raspberry Pi par USB pour la fonction tactile ; le boîtier prévoit donc le passage de ce câble ainsi que celui de l'alimentation. Des ouvertures sont également prévues pour favoriser le refroidissement du terminal embarqué.

La figure 3.4 présente les rendus mécaniques du boîtier. La figure 3.5 montre le boîtier imprimé ; cette photographie documente l'intégration mécanique, l'écran y étant éteint.

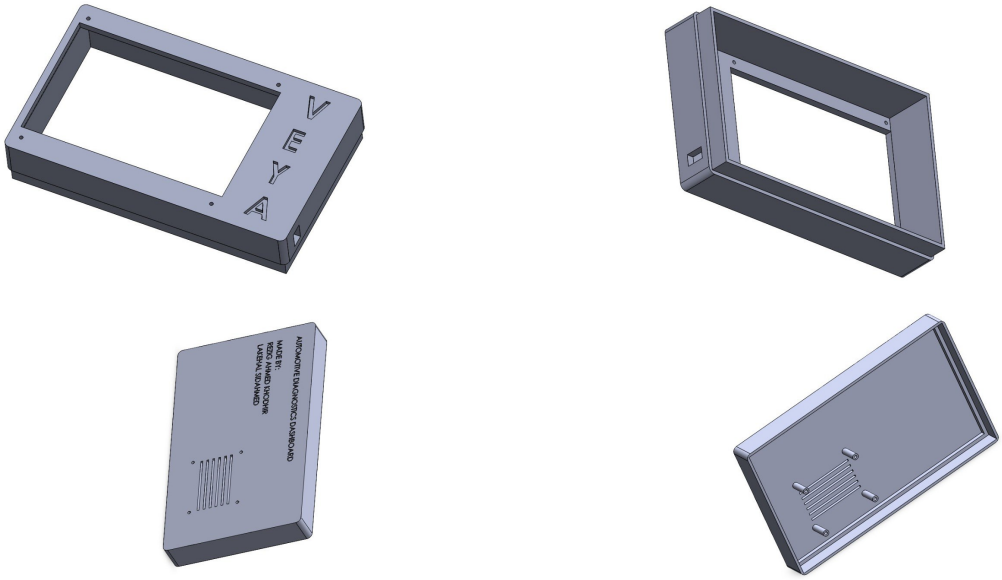


FIGURE 3.4 – Rendus SolidWorks du boîtier du tableau de bord embarqué VEYA.



FIGURE 3.5 – Boîtier imprimé du tableau de bord embarqué VEYA.

3.5 Conception du dispositif d'acquisition OBD

3.5.1 Rôle du dispositif d'acquisition

Le dispositif d'acquisition constitue l'interface locale entre le véhicule et le tableau de bord VEYA. Le prototype matériel est désormais assemblé et fonctionnel. Il prévoit deux voies d'acquisition : une entrée USB femelle destinée au câble OBDLink SX et une entrée DB9 destinée au câble OBD2–DB9 et à la voie CAN personnalisée. Il prépare ensuite les informations de télémétrie et de diagnostic avant de les transmettre sans fil au Raspberry Pi 5.

La communication est bidirectionnelle. Le dispositif transmet les données issues du véhicule et reçoit les commandes provenant du tableau de bord. Cette organisation permet de conserver une séparation nette entre la couche d'acquisition placée au niveau du port OBD-II et la couche de visualisation exécutée sur le terminal embarqué.

3.5.2 Architecture matérielle retenue

L'ESP32-S3 constitue le contrôleur embarqué principal du dispositif. Il coordonne l'acquisition des données, la gestion des commandes de diagnostic, la mise en forme des messages et la transmission BLE vers le Raspberry Pi. La figure 3.6 sert de repère pour les connexions du contrôleur ESP32-S3, notamment les lignes USB natives GPIO20/GPIO19 et les entrées/sorties utilisées pour la sélection de mode et les voyants.

La figure 3.7 présente les blocs fonctionnels de l'architecture retenue. Le schéma

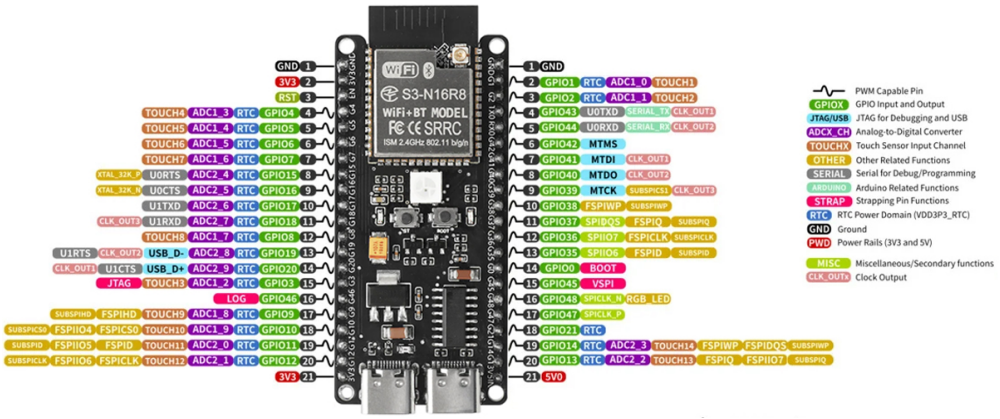


FIGURE 3.6 – Repérage des broches de la carte ESP32-S3 utilisée dans le prototype VEYA.

de câblage complète cette vue fonctionnelle en reportant les connexions précises des interfaces, de l'alimentation, du commutateur de mode et des voyants.

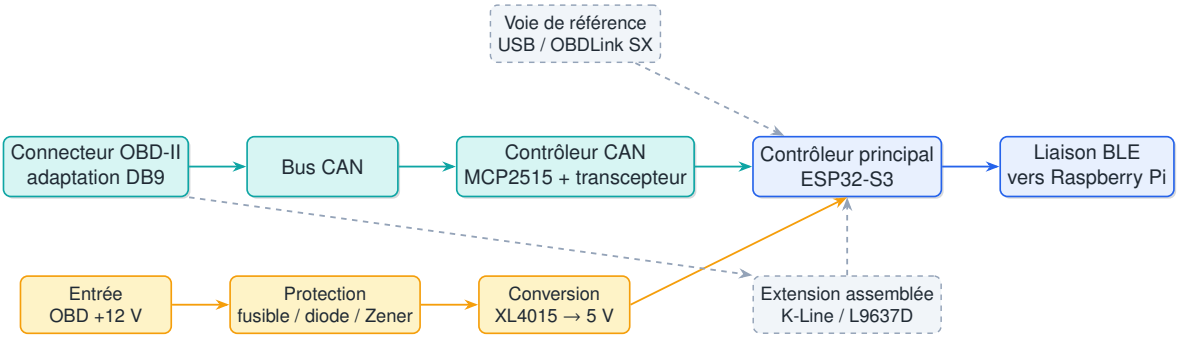


FIGURE 3.7 – Architecture fonctionnelle du dispositif d'acquisition OBD retenu pour VEYA.

3.5.3 Interfaces véhicule et acquisition CAN

Le raccordement au véhicule s'effectue par l'interface OBD-II. Pour la voie CAN personnalisée, l'architecture comprend une adaptation par connecteur DB9 et un module CAN fondé sur le circuit MCP2515, qui intègre son transcepteur sur la même carte. Cette organisation évite l'ajout d'un transcepteur discret au montage. Le module MCP2515 est entièrement câblé dans le prototype et communique avec le contrôleur principal par une liaison SPI.

Le brochage SPI retenu est le suivant : SCK de l'ESP32-S3 sur GPIO12 vers SCK du MCP2515, MOSI sur GPIO11 vers SI/MOSI, MISO sur GPIO13 vers SO/MISO, CS sur GPIO10 vers CS et INT sur GPIO9 vers INT. Le module MCP2515 est alimenté par le rail 5 V du prototype et sa masse est reliée à la masse commune. Côté véhicule, l'adaptation OBD-II/DB9 achemine les lignes CAN, la masse et le +12 V vers le dispositif. La voie CAN personnalisée est donc assemblée sur le plan matériel ; sa validation par le micrologiciel et sur véhicule reste cependant distincte de la voie USB/OBDLink SX déjà opérationnelle.

La même carte comporte une extension K-Line fondée sur le circuit L9637D. La ligne K issue de la broche 7 du connecteur OBD-II est acheminée, par l'adaptation DB9,

vers cette interface. Les connexions d'alimentation, de masse et de communication avec l'ESP32-S3 sont assemblées et vérifiées. Cette extension est matériellement achevée, mais elle n'a pas encore fait l'objet d'une campagne de validation sur véhicule avec le micrologiciel.

La figure 3.8 présente le brochage de référence OBD-II/DB9 utilisé comme support de câblage.

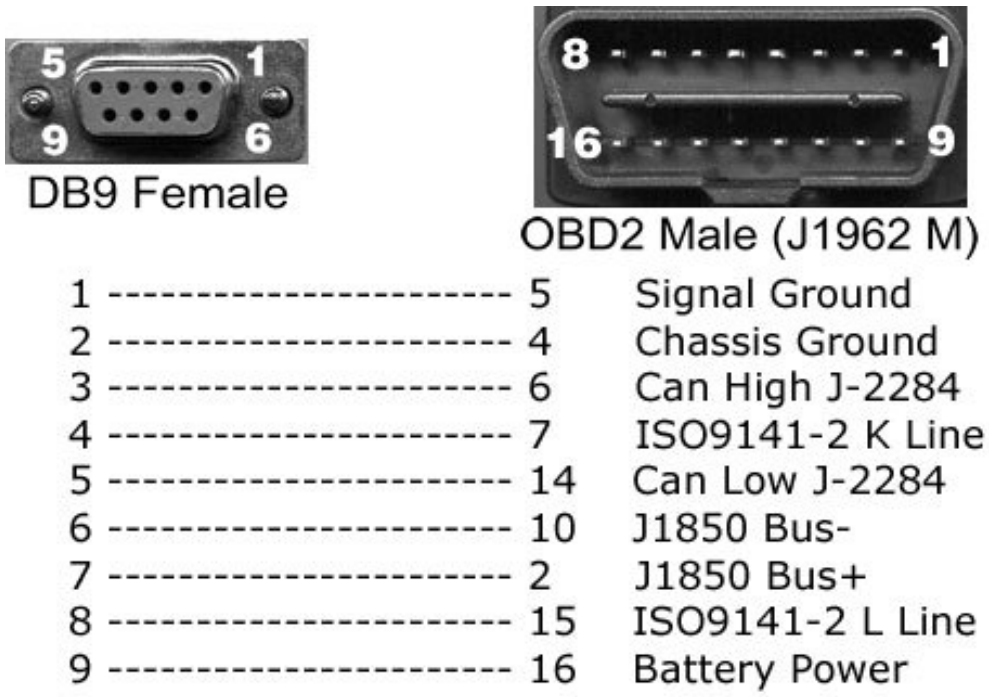


FIGURE 3.8 – Brochage de référence de l'adaptateur OBD-II vers DB9 utilisé pour le raccordement véhicule du dispositif VEYA.

L'entrée USB femelle associée au câble OBDLink SX est intégrée au dispositif comme voie d'acquisition principale validée. Son VBUS est alimenté par le rail 5 V régulé, sa ligne D+ est raccordée au GPIO20 de l'ESP32-S3, sa ligne D- au GPIO19 et sa masse à la masse commune. Le micrologiciel met en oeuvre le mode USB hôte, l'ouverture de l'interface FTDI/VCP de l'OBDLink SX, l'initialisation du protocole véhicule et la transmission des résultats vers le Raspberry Pi par BLE.

3.5.4 Alimentation et protection électrique

Le dispositif est alimenté depuis le +12 V arrivant par le connecteur DB9. Cette entrée traverse un fusible de 1 A, une diode série classique, puis un noeud de protection et de filtrage avant d'atteindre l'entrée du convertisseur abaisseur XL4015. Sur ce noeud, une diode Zener montée en inverse est raccordée à la masse comme branche d'écrêtage, avec deux condensateurs de filtrage de 470 µF et 100 µF vers la masse. La masse DB9 est raccordée directement à la masse d'entrée du XL4015 et à la masse commune du dispositif.

La sortie du XL4015 fournit le rail 5 V régulé du prototype. Ce rail alimente l'ESP32-S3, le module MCP2515, l'adaptateur USB femelle utilisé avec l'OBDLink SX et la LED rouge d'indication 5 V. La validation électrique de ce rail est présentée dans

le chapitre de réalisation ; dans la conception, il constitue l'alimentation commune des deux voies d'acquisition.

3.5.5 Sélection de mode et voyants

Un commutateur raccordé au GPIO4 de l'ESP32-S3 assure la sélection matérielle de la voie d'acquisition. Un niveau bas sélectionne le mode DB9/MCP2515, tandis qu'un niveau haut active le mode USB/OBDLink SX. Cette commande permet de changer de voie sans reprogrammer le dispositif.

Le prototype comprend quatre voyants dont le comportement est pris en charge par le micrologiciel final. La LED rouge, raccordée au rail 5 V, indique la présence de l'alimentation. La LED jaune sur GPIO5 signale le mode DB9/MCP2515, la LED bleue sur GPIO6 indique l'état de la liaison BLE et la LED verte sur GPIO7 signale le mode USB/OBDLink SX. Les essais d'intégration ont confirmé la cohérence entre la position du commutateur, la voie active et les indications lumineuses.

Après la description des interfaces véhicule, de l'alimentation, de la voie USB, du commutateur de mode et des voyants, la figure 3.9 synthétise le câblage complet du prototype.

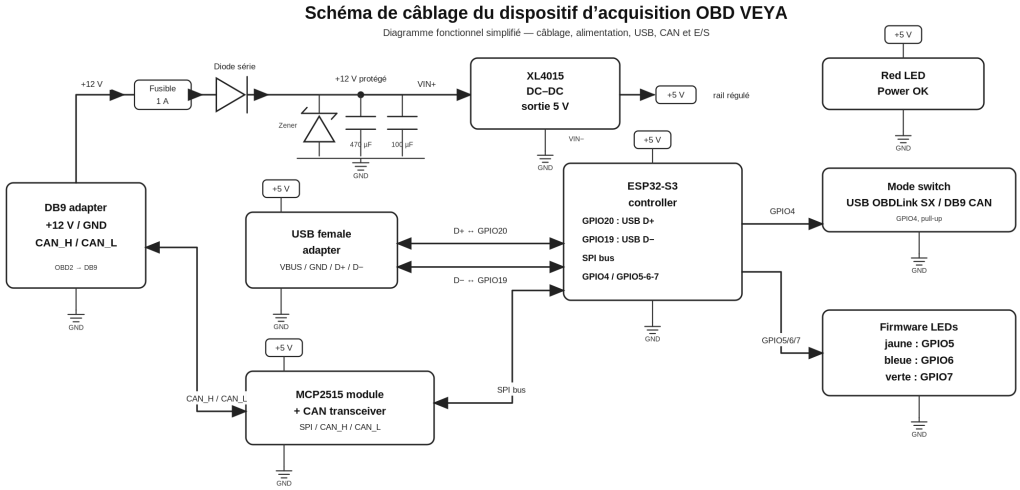


FIGURE 3.9 – Schéma fonctionnel de câblage du dispositif d'acquisition OBD VEYA, montrant la chaîne d'alimentation protégée, l'interface USB OBDLink SX, l'interface CAN via MCP2515 et les signaux de contrôle/état de l'ESP32-S3.

3.5.6 Communication et format des données

Le dispositif constitue le périphérique BLE de la liaison avec le Raspberry Pi. Il transmet les trames de télémétrie et les réponses de diagnostic au processus *bridge*, puis reçoit par le même canal les commandes destinées à la couche d'acquisition. Le choix et le fonctionnement détaillé de cette liaison sont présentés dans la section 3.6.

Les messages sont structurés au format JSON. Ils comprennent un champ `schema` destiné au versionnement, un champ `type` et les données associées. Cette organisation isole le transport BLE de la signification applicative des messages. Le modèle d'échange est présenté dans la section 3.6.

Le micrologiciel final regroupe dans une même application la gestion du commutateur et des voyants, l'ouverture USB hôte de l'OBDLink SX, l'acquisition des données vivantes et des DTC, ainsi que le transport BLE bidirectionnel vers le Raspberry Pi. Les interfaces MCP2515 et L9637D sont intégrées au prototype matériel, mais leur validation directe reste séparée de la voie USB actuellement exploitée.

3.5.7 Intégration mécanique du boîtier

Le dispositif d'acquisition est destiné à être intégré dans un boîtier dédié imprimé en 3D en PLA. Ce boîtier regroupe le contrôleur principal, le module CAN et l'étage d'alimentation, tout en maintenant l'accès aux connecteurs utiles et en permettant une installation adaptée à l'environnement du véhicule.

Le dimensionnement du boîtier s'appuie sur l'encombrement de la carte prototype, d'environ 9 cm × 15 cm. L'empreinte retenue du boîtier est de 9,5 cm × 15,5 cm, ce qui conserve une faible marge d'assemblage autour de la carte pour les connecteurs, le passage des câbles et la fixation interne. La hauteur du boîtier est comprise entre 2 cm et 3 cm.

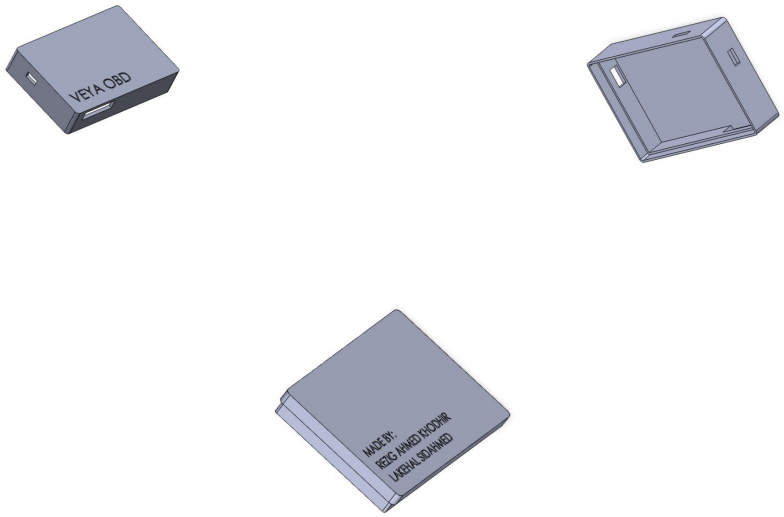


FIGURE 3.10 – Boîtier imprimé du dispositif d'acquisition OBD VEYA.

3.6 Choix de la communication entre dispositif OBD et Raspberry Pi

3.6.1 Besoin d'une liaison sans fil locale

Le dispositif d'acquisition est installé à proximité du port OBD-II, tandis que le Raspberry Pi 5 constitue le terminal utilisateur du tableau de bord. Une liaison sans

fil entre ces deux unités évite l'ajout d'un câblage de données dans l'habitacle et maintient leur indépendance fonctionnelle.

Cette séparation facilite l'intégration du dispositif au niveau du véhicule et permet au terminal d'assurer ses fonctions d'affichage sans dépendre de la position physique du connecteur OBD-II.

3.6.2 Justification du choix BLE

La liaison principale retenue est *Bluetooth Low Energy* (BLE). L'ESP32-S3 et le Raspberry Pi 5 disposent de cette capacité de communication. BLE fournit un canal bidirectionnel adapté à l'échange local des trames de télémétrie et des commandes de diagnostic, sans introduire de liaison filaire supplémentaire.

Le modèle de communication suit le principe d'un canal série de type *UART over BLE*, fondé sur le service Nordic UART. Dans le sens dispositif vers Raspberry Pi, les données sont transmises par notifications. Dans le sens inverse, les commandes sont envoyées vers une caractéristique de réception accessible en écriture. Les identifiants retenus sont les suivants :

— service NUS :
`6E400001-B5A3-F393-E0A9-E50E24DCCA9E;`
— caractéristique de notification, ESP32-S3 vers Raspberry Pi :
`6E400003-B5A3-F393-E0A9-E50E24DCCA9E;`
— caractéristique d'écriture, Raspberry Pi vers ESP32-S3 :
`6E400002-B5A3-F393-E0A9-E50E24DCCA9E.`

La caractéristique de notification transporte les messages émis par le dispositif, tandis que la caractéristique d'écriture transporte les commandes émises vers celui-ci.

Une variante fondée sur Bluetooth classique avec profil SPP est conservée comme solution de repli configurable. Elle ne constitue pas la liaison principale retenue pour l'architecture VEYA.

La figure 3.11 présente le principe de communication bidirectionnelle retenu entre le dispositif d'acquisition et le Raspberry Pi.

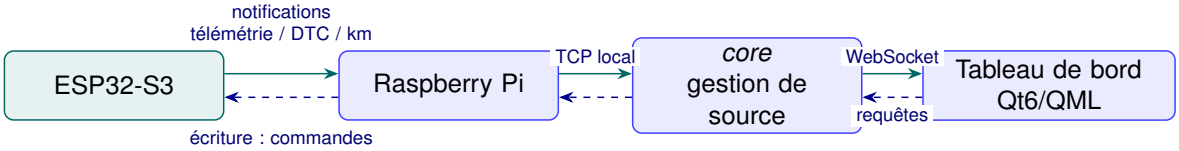


FIGURE 3.11 – Flux bidirectionnels entre le dispositif d'acquisition BLE et le tableau de bord Raspberry Pi.

3.6.3 Rôle du pont BLE sur Raspberry Pi

Sur le Raspberry Pi, le processus *bridge* isole la gestion de la liaison BLE du cœur de traitement. Il établit la connexion avec le dispositif ESP32-S3, reçoit les données transmises par notifications et reconstruit les messages complets avant de les relayer vers le *core* par une interface TCP locale.

Le pont assure également le chemin inverse. Les commandes issues du tableau de bord ou du *core* sont relayées vers le dispositif au moyen de la caractéristique BLE

accessible en écriture. Cette conception limite l'impact d'un changement de transport sur le reste de l'application.

3.6.4 Modèle de messages échangés

Les messages échangés sont structurés au format JSON. Ils comportent un champ `schema` destiné au versionnement et un champ `type` indiquant la nature du message. Les exemples suivants présentent uniquement les champs communs confirmés ; les charges utiles détaillées devront être complétées à partir du contrat d'interface final.

```
{"schema":1,"type":"telemetry","ts":7909572,"rpm":3599,
 "speed_kph":96.0,"coolant_c":85.1,"throttle_pct":54.0,
 "engine_load":72.1,"battery_v":13.65,"fuel_level":68.4,
 "intake_temp_c":30.1}

{"schema":1,"type":"query_dtc"}

{"schema":1,"type":"dtc",
 "codes":[{"code":"P0301","status":"stored"},
          {"code":"P0420","status":"pending"}]}

{"schema":1,"type":"query_mileage"}

{"schema":1,"type":"mileage_response","km":123456}
```

Cette valeur numérique est donnée comme exemple de format de charge utile et ne correspond pas à une mesure véhicule revendiquée. Le même modèle inclut la commande `clear_dtc` et la réponse `clear_dtc_result`. Cette organisation permet de distinguer les trames de télémétrie, les requêtes et les réponses de diagnostic sans lier leur sémantique au transport BLE.

La valeur courante du champ `schema` est `1`. Les trames de télémétrie regroupent les paramètres moteur sous forme de champs numérotés, notamment le régime, la vitesse, les températures, la charge, la position du papillon, la tension batterie et le niveau de carburant. Les réponses de diagnostic transportent une liste de codes, chaque code étant représenté par un objet associant l'identifiant du défaut et son statut, par exemple `stored` ou `pending`. La charge utile de la réponse kilométrage utilise le champ `km` au niveau du protocole. L'intégration complète jusqu'à l'affichage peut être traitée séparément lorsqu'elle est finalisée.

3.6.5 Considérations de fiabilité

La transmission BLE doit tenir compte du fait qu'un message JSON peut nécessiter une segmentation en plusieurs fragments. Les notifications reçues doivent donc être réassemblées avant le traitement du message complet. De même, les messages sortants doivent être sérialisés afin d'éviter l'entrelacement de fragments appartenant à des trames différentes. Les messages JSON sont délimités par un saut de ligne. Les écritures sortantes sont découpées selon la MTU disponible, tandis que les notifications entrantes sont accumulées dans un tampon jusqu'à reconstitution d'une ligne complète. Les écritures vers le dispositif sont traitées par un unique flux séquentiel, sans revendiquer une file explicite distincte.

La robustesse globale ne repose pas uniquement sur le transport. Le *core* gère l'état de la source active et applique le mécanisme de repli vers la source de démonstration lorsque le dispositif réel se déconnecte. La liaison BLE reste ainsi une couche de transport distincte de la logique de sélection de source.

3.7 Conception de la chaîne d'acquisition et de diagnostic

3.7.1 Nature des données acquises

La chaîne VEYA prend en charge plusieurs catégories de données issues du véhicule. La télémétrie en temps réel comprend le régime moteur, la vitesse, la température du liquide de refroidissement, la position du papillon, la charge moteur, la tension de la batterie, le niveau de carburant et la température d'admission.

La chaîne traite également les informations de diagnostic : codes défaut avec leur statut, lecture du kilométrage et réponses aux commandes contrôlées. Des informations d'état et d'avertissement complètent ces données afin d'alimenter l'interface locale.

3.7.2 Chaîne locale d'acquisition et de diffusion

La chaîne locale commence au niveau des calculateurs du véhicule accessibles par l'interface OBD-II et le bus CAN. Le dispositif ESP32-S3 acquiert les données et transforme les résultats utiles en messages structurés. Ces messages sont transmis par BLE au processus *bridge* exécuté sur le Raspberry Pi 5.

Le pont relaie les messages vers le *core*. Ce service central assure le routage des données, la sélection de la source active et la transmission vers le service *ws_server*. La diffusion locale par WebSocket met ensuite les informations à disposition du tableau de bord Qt6/QML.

La figure 3.12 présente le flux principal des données ainsi que le chemin inverse emprunté par les commandes.

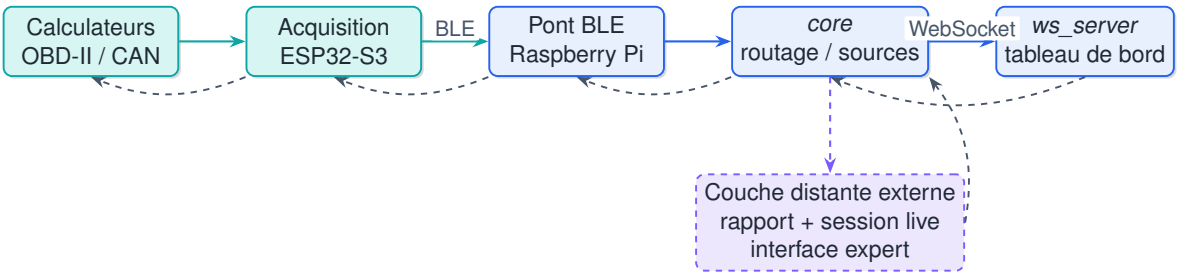


FIGURE 3.12 – Chaîne locale d'acquisition et flux de diagnostic de VEYA, complétés par la couche distante externe.

3.7.3 Machine d'états de sélection de source

Le *core* gère automatiquement la provenance des données au moyen d'une machine d'états. Lorsque le dispositif réel n'est pas connecté, une source de démonstration

fournit des données simulées. Le système maintient alors un état d'attente de connexion du dispositif.

Lorsque le pont établit la connexion avec l'ESP32-S3, il demande la prise en charge de la source active. Le *core* libère la source de démonstration et bascule vers les données réelles. Si le dispositif se déconnecte, la source de démonstration est relancée automatiquement. Un mode de démonstration forcé peut également être sélectionné par l'utilisateur. Cette machine d'états constitue un principe de robustesse de l'architecture locale. Les deux transitions automatiques ont été vérifiées : passage de la source de démonstration à la source réelle lors de la connexion, puis retour vers la source de démonstration lors de la déconnexion.

3.7.4 Flux des commandes de diagnostic

Le protocole applicatif limite les commandes transmises au dispositif d'acquisition. Une demande issue du tableau de bord, ou de l'interface expert pendant une session distante, est remise au *core*, relayée par le *bridge*, puis transmise à l'ESP32-S3. Le dispositif formule la requête destinée au véhicule et renvoie la réponse par la même chaîne.

Les commandes prévues comprennent `query_dtc` pour la lecture des codes défaut, `clear_dtc` pour leur effacement contrôlé et `query_mileage` pour la lecture du kilométrage. Cette circulation bidirectionnelle conserve une interface commune entre l'utilisateur local, l'expert distant et le dispositif embarqué.

3.7.5 Couche distante de rapport et session live

La chaîne locale expose les données et les commandes nécessaires à une couche logicielle distante. Cette couche est intégrée au système et sa validation de bout en bout est présentée au chapitre 4. Elle reste cependant externe au périmètre embarqué constitué par le dispositif OBD et le tableau de bord Raspberry Pi.

Pour la génération de rapport, les DTC sont transmis au serveur Django associé à une base PostgreSQL. Le service retourne une information structurée comprenant la signification des codes, les causes probables, les symptômes et les réparations recommandées. Pour la session live, un relais indépendant permet à l'expert d'observer la télémétrie depuis un navigateur et de demander une lecture des DTC à travers la chaîne de commandes.

3.7.6 Enrichissement constructeur Renault/Dacia par XML DDT2000

Les DTC standard peuvent être interprétés à partir de la base de diagnostic existante. Pour des besoins Renault et Dacia plus approfondis, la chaîne peut être enrichie par l'exploitation de fichiers XML issus de DDT2000. Cette opération ne doit pas être exécutée dans l'ESP32-S3 : elle appartient à la couche logicielle de diagnostic côté serveur ou à l'environnement de développement.

L'enrichissement peut s'appuyer sur le contexte du calculateur, les identifiants, les DTC bruts, les identifiants de données et les correspondances XML candidates. Une correspondance issue d'un fichier XML ne doit toutefois pas être considérée comme correcte sans vérification empirique. Cette discipline de preuve s'applique en particulier aux interprétations constructeur.

Le contrat de données transmis vers la couche d'enrichissement conserve les éléments bruts nécessaires à une interprétation ultérieure vérifiable : le contexte du calculateur interrogé, les identifiants de données, les codes de défaut bruts et les réponses telles que reçues. Le principe directeur consiste à transmettre l'intégralité des données disponibles sans interprétation préalable côté dispositif, la couche d'enrichissement sélectionnant ensuite les éléments pertinents. Cette organisation préserve la traçabilité et évite d'inscrire une interprétation non vérifiée dans la chaîne d'acquisition.

3.7.7 Encadrement des opérations de diagnostic

La chaîne locale est conçue pour exécuter un ensemble limité de commandes de diagnostic. L'effacement des DTC est traité comme une opération contrôlée. Les opérations susceptibles de modifier arbitrairement l'état d'un calculateur, notamment l'écriture libre, l'accès sécurisé ou le contrôle de routines, sont exclues du flux local automatique.

Les données brutes et les réponses de diagnostic doivent être conservées afin d'assurer la traçabilité des interprétations et de permettre leur vérification ultérieure.

La journalisation est assurée côté Raspberry Pi : les processus de la chaîne locale écrivent des journaux distincts dans un répertoire dédié, ce qui permet de conserver les trames brutes et les réponses de diagnostic reçues. Cette conservation séparée des journaux soutient la traçabilité des interprétations et autorise leur vérification a posteriori, conformément à la discipline de preuve adoptée dans le projet.

3.8 Conclusion

Ce chapitre a présenté l'architecture retenue pour VEYA, organisée autour du dispositif d'acquisition relié à l'interface OBD-II, de la liaison BLE et du tableau de bord exécuté sur Raspberry Pi 5. Cette chaîne embarquée assure l'acquisition, la transmission et la restitution locale des données de télémétrie et de diagnostic.

La conception repose sur une séparation explicite entre la couche d'acquisition, la couche de transport, le cœur de traitement et la couche de présentation. Une couche logicielle distante complète cette organisation pour la génération de rapports et les sessions de diagnostic à distance, tout en restant externe au système embarqué local.

Le dispositif OBD est entièrement assemblé et son micrologiciel principal est opérationnel. Il prend en charge la voie USB hôte avec l'OBDLink SX, la communication BLE, les commandes de diagnostic, le changement de mode et les indicateurs lumineux. La voie CAN fondée sur le MCP2515 ainsi que l'extension K-Line fondée sur le L9637D sont ègalement assemblées sur le plan matériel ; leurs essais spécifiques par micrologiciel et sur véhicule sont toutefois conservés comme validations distinctes.

Cette organisation modulaire permet d'assurer l'usage local du tableau de bord et d'exposer les données et commandes nécessaires au flux de diagnostic distant. Le chapitre suivant présente la mise en œuvre de cette architecture et la validation expérimentale de la chaîne complète, depuis le véhicule jusqu'au rapport et à l'interface de l'expert.

CHAPITRE 4

Mise en œuvre expérimentale et validation

4.1 Introduction

Ce chapitre présente la mise en œuvre pratique et la validation expérimentale de la chaîne d'acquisition et du tableau de bord VEYA. La démarche adoptée repose sur une progression par étapes, depuis les premiers essais avec des adaptateurs ELM327 Bluetooth jusqu'à l'analyse plus approfondie des protocoles OBD et CAN, puis à l'intégration de la chaîne locale.

Les essais ont été conduits sur des véhicules réels afin de distinguer les fonctions accessibles par l'OBD standard, les limites propres aux interfaces utilisées et les besoins à prendre en compte pour le dispositif d'acquisition retenu. Les résultats sont présentés en conservant une distinction explicite entre les observations confirmées et les interprétations qui nécessitent encore une validation empirique.

4.2 Mise en œuvre expérimentale de la chaîne de diagnostic

La mise en œuvre expérimentale a été conduite sur plusieurs véhicules réels afin d'observer le comportement de la chaîne de diagnostic dans des contextes protocolaires différents. Les premiers essais ont utilisé un adaptateur ELM327 Bluetooth clone comme interface de référence. Des scripts Python fondés sur la bibliothèque *pyserial* ont permis d'envoyer les commandes, d'enregistrer les réponses et d'analyser progressivement les données obtenues. Lorsqu'il était disponible, un scanner professionnel Launch a fourni un point de comparaison pour distinguer les informations accessibles par l'OBD standard des données propres aux outils de diagnostic constructeur.

Cette démarche a permis d'examiner successivement l'identification du protocole de communication, la lecture de paramètres en temps réel, la lecture des codes défaut et les limites liées aux réponses comportant plusieurs trames. Les essais ont également mis en évidence la différence entre l'accès standard OBD-II et l'accès à des fonctions constructeur, ainsi que la nécessité de prendre en compte les contextes CAN et KWP2000 dans l'analyse expérimentale.

La progression retenue comprend cinq phases :

1. essais initiaux avec des adaptateurs ELM327 Bluetooth ;
2. transition vers des interfaces OBD plus fiables ;
3. exploration approfondie des protocoles et outils Renault/Dacia ;

4. intégration de la chaîne locale VEYA ;
5. validation du prototype matériel retenu.

Les interfaces plus performantes, notamment le câble OBDLink SX, ainsi que les outils RenoLink, DDT2000 et Wireshark sont introduits dans les sections suivantes. L'objectif de cette progression est de transformer les observations expérimentales en exigences concrètes pour le dispositif d'acquisition et pour son raccordement au tableau de bord Raspberry Pi.

4.3 Premiers essais avec adaptateurs ELM327 Bluetooth

4.3.1 Objectif de la phase ELM327

La première phase expérimentale avait pour objectif d'évaluer les possibilités offertes par un adaptateur ELM327 Bluetooth clone dans des situations réelles. Cette interface a servi à vérifier la lecture des services OBD-II standard et à identifier les limites susceptibles d'influencer la conception de la chaîne VEYA.

Les essais ont porté sur quatre véhicules de constructeurs et de générations différents. Ils ont permis d'observer deux familles de protocoles, KWP2000 sur ligne K et CAN, ainsi que plusieurs restrictions liées au transport multi-trames et à l'accès aux fonctions constructeur.

4.3.2 Véhicules et conditions d'essai

Les sessions documentées concernent une Renault Symbol 2010, une Toyota Yaris 2012, un Citroën Berlingo 2016 et une Hyundai Accent 2020. Les commandes ont été envoyées au moyen de scripts Python et les réponses ont été enregistrées sous forme de journaux. Le scanner Launch a été utilisé comme outil de comparaison afin d'identifier les informations qui restaient hors de portée de l'accès OBD standard.

La figure 4.1 illustre une partie des véhicules utilisés durant cette première phase d'essais. Ces photographies documentent le contexte expérimental sans constituer, à elles seules, une preuve des résultats techniques présentés ci-après.



FIGURE 4.1 – Exemples de véhicules utilisés lors de la phase initiale d'essais avec adaptateur ELM327 Bluetooth : Renault Symbol 2010, Citroën Berlingo 2016 et Hyundai Accent 2020.

4.3.3 Résultats par véhicule

Renault Symbol 2010

L'essai sur Renault Symbol 2010 a conduit à l'identification du protocole ISO 14230-4 KWP2000 sur ligne K. Après prise en compte de l'en-tête propre aux trames KWP, les données live OBD standard ont pu être lues. La lecture du NIV a également été obtenue par réassemblage des trames successives.

La comparaison avec le scanner Launch a toutefois montré que les fonctions OBD standard ne donnaient pas accès à l'ensemble des codes constructeur et des calculateurs du véhicule. Cette session a mis en évidence la nécessité de prévoir, pour une version universelle du dispositif VEYA, une prise en compte future de la ligne K et du protocole KWP2000.

Toyota Yaris 2012

La Toyota Yaris 2012 a permis de confirmer l'utilisation du protocole ISO 15765-4 CAN avec identifiants sur 11 bits et un débit de 500 kbps. Le nombre de paramètres OBD accessibles était plus important que sur la Symbol. Les lectures obtenues comprenaient notamment les paramètres moteur usuels, le débit massique d'air et la pression barométrique.

La lecture du NIV n'a pas abouti : la réponse multi-trames n'a pas été complétée par le clone ELM327. Par ailleurs, la requête standard Mode 03 ne faisait pas apparaître toutes les informations de défaut internes visibles avec les outils de comparaison. Ce résultat a confirmé que l'accès OBD standard ne suffit pas à couvrir l'ensemble des besoins de diagnostic.

Citroën Berlingo 2016

Le Citroën Berlingo 2016 a également confirmé l'utilisation du protocole CAN 11 bits à 500 kbps. Cette session a montré l'intérêt des paramètres OBD étendus propres au contexte diesel, notamment pour l'observation de la pression de rail, des informations relatives au turbocompresseur et de plusieurs paramètres de post-traitement.

Les enregistrements réalisés pendant la conduite ont permis d'identifier des observations cohérentes avec des anomalies de capteurs, sans généraliser ces interprétations au-delà des données enregistrées. L'essai a aussi mis en évidence les restrictions imposées par la passerelle PSA et par les sessions propriétaires : l'accès aux calculateurs autres que le calculateur moteur et aux diagnostics constructeur est resté limité.

Hyundai Accent 2020

La Hyundai Accent 2020 a confirmé le protocole CAN 11 bits à 500 kbps et une disponibilité importante de paramètres OBD standard et étendus. Pour la lecture du NIV, la première trame contenait les caractères `MAL`, conformes au début du NIV du véhicule, mais la réponse complète n'a pas été reconstituée en raison de la limitation multi-trames du clone.

Les relevés effectués après réchauffement du moteur ont également permis de distinguer une lecture transitoire observée au démarrage d'une observation

persistante concernant un capteur. Cette session confirme l'importance de documenter les conditions d'acquisition avant d'interpréter une valeur comme une anomalie.

4.3.4 Synthèse des limites observées

Le tableau 4.1 rassemble les principaux enseignements tirés des quatre sessions. Il ne reprend que les observations utiles à l'évolution de la chaîne VEYA.

Véhicule	Protocole	Interface	Lectures principales obtenues	Limite principale	Enseignement pour VEYA
Renault Symbol 2010	ISO 14230-4 KWP2000	ELM327 Bluetooth clone	Données live OBD standard et NIV multi-trames	Accès limité aux codes standard et au calculateur moteur	Prévoir la prise en compte future de la ligne K pour une couverture plus large
Toyota Yaris 2012	ISO 15765-4 CAN 11 bits, 500 kbps	ELM327 Bluetooth clone	Paramètres moteur, débit massique d'air et pression barométrique	NIV incomplet et informations constructeur absentes du Mode 03 standard	Maîtriser le transport multi-trames et distinguer OBD standard et diagnostic constructeur
Citroën Berlingo 2016	ISO 15765-4 CAN 11 bits, 500 kbps	ELM327 Bluetooth clone	Paramètres live et données diesel étendues utiles pendant la conduite	Passerelle PSA et sessions propriétaires limitant l'accès hors moteur	Prendre en compte les restrictions propres au constructeur
Hyundai Accent 2020	ISO 15765-4 CAN 11 bits, 500 kbps	ELM327 Bluetooth clone	Nombreux paramètres live et premiers caractères du NIV	NIV incomplet en raison de la gestion multi-trames du clone	Conserver les conditions d'acquisition et maîtriser le contrôle de flux

Tableau 4.1 – Synthèse des premiers essais réalisés avec un adaptateur ELM327 Bluetooth clone.

4.3.5 Apports pour l'architecture VEYA

Cette première campagne a confirmé l'intérêt d'une approche multi-véhicules et progressive. L'adaptateur ELM327 Bluetooth a permis de mettre en place les scripts initiaux, de lire des données live et de caractériser les différences entre KWP2000 et CAN. Il a toutefois montré des limites pour le réassemblage de certaines réponses multi-trames, pour l'observation fiable du bus et pour l'accès aux couches constructeur.

Ces constats ont motivé le recours à des interfaces plus performantes pour la suite de l'analyse. Ils ont également guidé les exigences de la chaîne VEYA : séparation des couches protocolaires, traitement explicite des trames multiples, conservation des données brutes et prudence dans l'interprétation des codes de défaut.

4.4 Transition vers des interfaces plus performantes

4.4.1 Limites observées avec les interfaces Bluetooth génériques

Les adaptateurs ELM327 Bluetooth clones ont constitué un point de départ utile pour l'apprentissage des commandes OBD-II standard et pour la conduite des premières sessions sur véhicules réels. Ils ont permis de confirmer la lecture de données live dans des contextes KWP2000 et CAN, tout en fournissant une référence comparative pour les étapes suivantes.

Les essais ont cependant mis en évidence plusieurs limites. L'accès aux diagnostics propres aux constructeurs est resté restreint, notamment lorsque des sessions spécifiques ou des calculateurs autres que le calculateur moteur devaient être interrogés. Sur les véhicules CAN, la gestion des réponses multi-trames et du contrôle de flux s'est également révélée insuffisante dans certains cas. Enfin, l'observation passive du trafic a produit des données dont l'exploitation était limitée par la qualité de la restitution fournie par le clone.

4.4.2 Intérêt des interfaces USB et de l'OBDLink SX

Ces constats ont motivé l'adoption d'interfaces USB plus fiables pour l'analyse approfondie. Le câble OBDLink SX a été retenu comme interface principale pour cette phase. Il permet de travailler avec une liaison filaire stable et de mieux maîtriser l'envoi des commandes ainsi que la collecte des réponses nécessaires à l'étude des protocoles constructeur.

L'ELM327 Bluetooth n'a pas été abandonné : il reste utile comme interface de référence pour les fonctions OBD-II standard et pour la comparaison des comportements. Il ne constitue toutefois pas, à lui seul, une base suffisante pour l'analyse avancée des véhicules Renault et Dacia.

4.4.3 Conséquences sur la méthode de développement

La transition vers l'interface USB ne correspond pas à un simple remplacement matériel. Elle marque un changement de méthode : les sessions de diagnostic doivent être reproductibles, les requêtes et réponses doivent être conservées, et les interprétations doivent pouvoir être reliées aux trames brutes observées.

Cette approche conduit à associer l'interface OBDLink SX à des scripts contrôlés et à des outils de capture. Elle prépare la mise au point de la chaîne VEYA en distinguant explicitement la couche physique, le transport des messages, leur interprétation et leur normalisation pour le tableau de bord.

4.5 Analyse approfondie des protocoles et outils Renault/Dacia

4.5.1 Complémentarité des outils spécialisés

L'analyse Renault/Dacia ne repose pas sur un outil unique. Elle combine des interfaces de diagnostic, des outils logiciels, des captures de trafic et des scripts Python afin de confronter plusieurs sources d'information. Cette démarche est nécessaire pour distinguer les services OBD-II standard des requêtes propres à une famille de calculateurs.

Les outils DDT4All et PyRen permettent d'explorer les calculateurs Renault et Dacia et de situer les requêtes disponibles dans leur contexte fonctionnel. RenoLink constitue un autre point d'observation des échanges de diagnostic. Les fichiers XML issus de la base DDT2000 fournissent une description structurée des familles de calculateurs et des données associées. Wireshark est utilisé pour la capture et le filtrage des trames, tandis que les scripts Python permettent de reproduire des requêtes ciblées, de conserver les réponses et d'automatiser leur analyse.

Les essais ont été menés avec une chaîne logicielle fondée sur des scripts Python dédiés à la communication série et à l'analyse des journaux, complétée par des outils de capture et d'inspection des trames. Les versions précises ne sont pas détaillées ici, l'objectif étant de décrire la démarche expérimentale plutôt que l'environnement logiciel.

4.5.2 Capture et analyse des échanges

Les outils de diagnostic génèrent des couples requête–réponse entre le poste d'analyse et les calculateurs du véhicule. La capture de ces échanges permet d'observer l'ordre des messages, leur adressage et les réponses retournées par les calculateurs. Wireshark fournit un support de filtrage et de lecture des traces enregistrées.

Cette observation ne constitue pas, à elle seule, une preuve de la signification d'une trame. Elle permet en revanche de reconstruire la logique protocolaire, de comparer le comportement d'un outil existant avec celui d'un script ciblé et de préparer l'implémentation future dans le dispositif d'acquisition VEYA. Les journaux bruts et les captures doivent être conservés afin d'assurer la traçabilité des interprétations.

4.5.3 Exploitation des fichiers XML DDT2000

Les fichiers XML DDT2000 décrivent des familles de calculateurs Renault et Dacia. Pour l'analyse Dacia Duster, les fichiers suivants sont signalés comme effectivement exploités :

— `SIRIUS32_EMS3130_Soft_REB_460_20100610T153641.xml`;
— `UCH_transv_7_1_20130122T095956.xml`.

Un fichier XML peut contenir des métadonnées de cible, des informations d'identification automatique (*AutoIdent*), des requêtes, des définitions de données ainsi que des écrans et libellés destinés aux outils de diagnostic. Ces informations permettent de rechercher des calculateurs candidats, de comprendre les requêtes disponibles et d'identifier des pistes pour le décodage des données live, des identifiants et des codes défaut constructeur.

Certains fichiers peuvent également décrire des opérations avancées, notamment des fonctions de configuration, de codage ou de commande de routines. Ces opérations sont exclues du flux automatique de VEYA. De même, une correspondance extraite d'un fichier XML reste une hypothèse de travail tant qu'elle n'a pas été confirmée empiriquement sur les données observées.

4.5.4 Positionnement des protocoles

L'OBD-II définit un ensemble de services standard destinés notamment à la lecture de paramètres moteur et de codes défaut. Ces services peuvent être transportés sur plusieurs couches de communication. Dans un contexte CAN, le bus assure l'échange des trames au niveau physique et liaison de données. Des services UDS peuvent être utilisés au-dessus de cette couche pour des fonctions de diagnostic plus avancées. Lorsque la taille d'un message dépasse celle d'une trame CAN, ISO-TP prend en charge sa segmentation et son réassemblage [12, 13].

Pour certains véhicules Renault et Dacia plus anciens, l'analyse doit également prendre en compte KWP2000 sur ligne K. La fiche expérimentale du Duster signale l'étude d'une séquence KWP2000 avec initialisation rapide (*FastInit*) [11].

Les journaux expérimentaux du Duster confirment la séquence de préparation suivante : `ATWS`, `ATE0`, `ATL0`, `ATS0`, `ATAL`, `STPTO 500`, `STP 25`, `ATSW 96` et `ATTA F1`. Le message de réveil `ATWM` et l'en-tête `ATSH` sont ensuite adaptés à l'adresse du calculateur interrogé. L'envoi de la requête de session `10 C0` a produit une réponse positive lors des sessions *FastInit* confirmées.

4.5.5 Chaîne de traitement expérimentale

Les scripts Python constituent le lien entre l'observation des outils existants et l'intégration dans VEYA. Ils envoient les commandes vers l'interface, journalisent les réponses, extraient les trames brutes et réassemblent les messages segmentés lorsque le protocole l'exige. Les informations obtenues peuvent ensuite être confrontées aux fichiers XML DDT2000 afin de classer les calculateurs candidats et de rechercher les définitions pertinentes.

Les données retenues sont finalement normalisées avant leur exploitation par la chaîne VEYA. La figure 4.2 résume ce flux de travail sans préjuger de la validité d'une interprétation particulière.

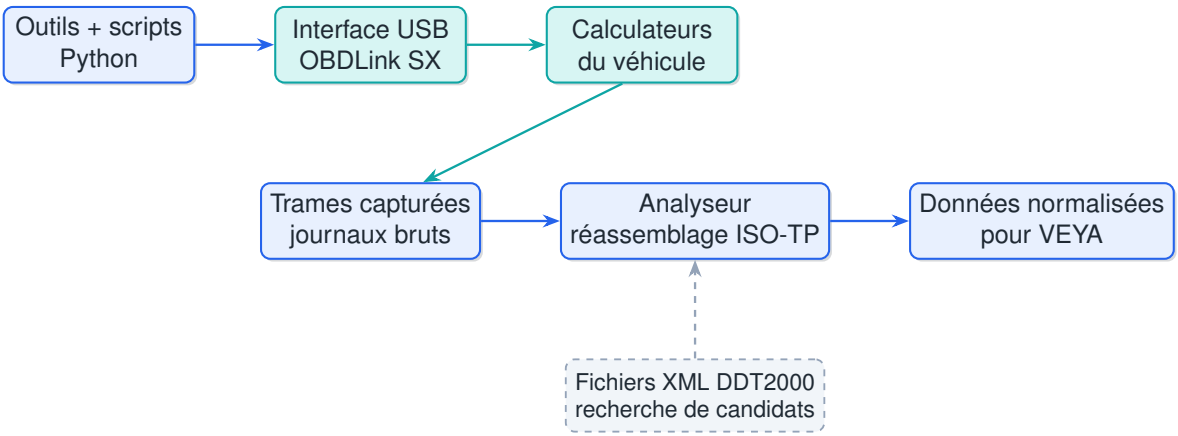


FIGURE 4.2 – Chaîne expérimentale d'analyse des outils et protocoles Renault/Dacia.

4.5.6 Encadrement des opérations et discipline de preuve

L'analyse expérimentale est limitée aux opérations nécessaires à la compréhension de la chaîne de diagnostic. Les écritures arbitraires dans les calculateurs, les accès de sécurité, les fonctions de codage et les commandes de routines sont exclus du flux automatique. Une capacité d'écriture éventuellement observée dans des trames ou dans un fichier XML ne doit donc pas être présentée comme validée sur véhicule.

La même prudence s'applique au décodage des DTC constructeur. Une association entre un code, un bit et un libellé XML doit être confirmée par des observations empiriques, par exemple par effacement puis nouvelle capture dans des conditions contrôlées. La conservation des traces brutes permet de documenter cette démarche et d'éviter de transformer une hypothèse de décodage en résultat acquis.

4.6 Résultats obtenus sur Renault Symbol 2019

4.6.1 Objectif de l'essai sur Renault Symbol 2019

La Renault Symbol 2019 a constitué le principal cas de validation de la chaîne CAN/UDS mise en place dans le cadre de ce travail. L'objectif était de vérifier l'acquisition de données de diagnostic sur plusieurs calculateurs, l'extraction des DTC, la lecture du kilométrage et l'exploitation complémentaire de captures PCAP. L'analyse

hors ligne a également permis d'étudier l'apport des fichiers XML DDT2000 pour enrichir les données brutes sans confondre la mesure observée avec son interprétation.

4.6.2 Identification des contextes ECU

Les essais en lecture seule ont permis d'identifier plusieurs contextes de communication CAN. Le tableau 4.2 distingue les contextes confirmés lors des lectures sur véhicule de ceux exploités dans les captures PCAP et dans la couche d'enrichissement.

Tableau 4.2 – Contextes ECU exploités pour la Renault Symbol 2019

ECU / fonction	ID requête	ID réponse	Rôle dans l'essai	Type de preuve
Injection / moteur	`7E0`	`7E8`	Lecture des DTC moteur et analyse des transactions UDS	Live et PCAP V2
Tableau de bord / cluster	`743`	`763`	Lecture du kilométrage et analyse des DTC du tableau de bord	Live pour le kilométrage ; PCAP V2 et enrichissement pour les DTC
Airbag	`752`	`772`	Vérification d'une lecture DTC en mode non destructif	Live

L'analyse de la capture V2 a aussi fait apparaître les contextes `740 -> 760`, `8101F1 -> 7E8`, `8151F1 -> 760` et `817AF1 -> 763`. Ils sont conservés dans les traces pour assurer la traçabilité, mais ne sont pas attribués ici à une fonction ECU sans preuve suffisante.

4.6.3 Extraction des DTC et interprétation

La capture PCAP V2, enregistrée avant l'effacement des défauts, a permis d'extraire les DTC présentés dans le tableau 4.3. Les journaux expérimentaux VEYA montrent que les valeurs brutes, les codes normalisés pour l'affichage et les libellés d'interprétation doivent rester explicitement séparés.

Tableau 4.3 – DTC extraits de la capture PCAP V2 de la Renault Symbol 2019

ECU	DTC brut	Statut	Code affiché	Libellé issu de l'enrichissement DDT2000	Source de preuve
Injection / moteur	`0226F1`	`28`	`P0226-F1`	`Invalid Data Received From Cluster Sensor`	PCAP V2 ; enrichissement
Injection / moteur	`1525F3`	`28`	`P1525-F3`	Libellé spécifique non confirmé	PCAP V2 ; affichage générique
Tableau de bord / cluster	`94037B`	`A8`	`9403`	`Brake_Fluid_Level`	PCAP V2 ; enrichissement
Tableau de bord / cluster	`F00316`	`28`	`F003`	`Power Supply monitoring`	PCAP V2 ; enrichissement
Tableau de bord / cluster	`C00187`	`A8`	`C001`	`Lost CAN communication`	PCAP V2 ; enrichissement

Dans cette même capture, aucun DTC airbag n'a été observé. Lors de la lecture live finale, la réponse airbag était valide et ne contenait aucun DTC après un effacement réalisé antérieurement. La lecture live du calculateur moteur a, quant à

elle, retourné la valeur brute `C12187` avec le statut `28`, affichée sous la forme `U0121-87`. Ces résultats ne signifient pas que chaque libellé enrichi est universellement valide : les correspondances doivent rester vérifiées empiriquement selon le contexte du calculateur et du véhicule.

4.6.4 Extraction du kilométrage

La lecture live du tableau de bord a permis d'extraire le kilométrage à partir du DID `0207`. La valeur brute `02360C` a été décodée en `144908 km`. Ce résultat a été comparé à la valeur observée sur le véhicule et au diagnostic de référence. La même valeur apparaît dans la capture PCAP V2, ce qui fournit un point de comparaison entre l'acquisition live et le rejeu hors ligne.

Cette observation valide une capacité de lecture. Aucune fonction d'écriture du kilométrage n'a été implémentée ni validée sur véhicule dans le cadre de ce travail.

4.6.5 Apport de l'enrichissement DDT2000

Les octets issus des transactions UDS ne suffisent pas toujours à produire une information directement exploitable par l'utilisateur. L'indexation des fichiers XML DDT2000 a servi de couche d'enrichissement pour rechercher des candidats correspondant aux contextes ECU, associer certains DTC constructeur à des libellés et interpréter le DID de kilométrage.

Cette couche reste distincte de l'acquisition brute. Les fichiers candidats sont classés, les valeurs sources sont conservées et les interprétations ne sont retenues qu'avec un niveau de preuve suffisant. L'essai sur Symbol 2019 ne constitue donc pas une validation exhaustive de l'ensemble des libellés XML disponibles.

4.6.6 Bilan de validation

Le cas Renault Symbol 2019 a permis de valider la communication CAN/UDS avec des contextes ECU identifiés, la lecture de DTC en mode non destructif et l'extraction du kilométrage par DID. Il a également confirmé l'intérêt du rejeu PCAP pour reproduire l'analyse hors ligne et de l'enrichissement DDT2000 pour rendre certains résultats plus lisibles.

Cette validation conduit à maintenir trois niveaux clairement séparés dans VEYA : la donnée brute conservée pour la traçabilité, la normalisation destinée à l'affichage et l'interprétation enrichie qui doit être corroborée par l'expérimentation. Les opérations destructives, notamment l'écriture de DID ou du kilométrage, ne font pas partie du flux validé.

4.7 Résultats obtenus sur Dacia Duster 2012

4.7.1 Objectif de l'essai sur Dacia Duster 2012

La Dacia Duster 2012 a constitué un cas d'investigation complémentaire à la Renault Symbol 2019. Alors que cette dernière a permis de valider une chaîne CAN/UDS, le Duster a mis en évidence un contexte de diagnostic plus ancien

fondé sur KWP2000 et la ligne K. Ce véhicule a donc servi à étudier les limites d'une approche universelle et à préciser l'intérêt de la prise en charge KWP2000/K-Line dans VEYA et de guider la réalisation de l'extension L9637D assemblée sur le prototype.

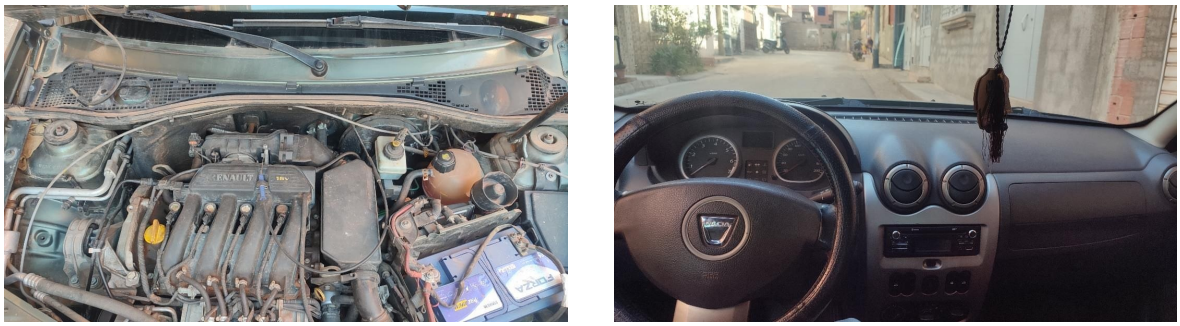


FIGURE 4.3 – Contexte expérimental sur Dacia Duster 2012 : compartiment moteur et montage d'essai.

4.7.2 Mise en évidence du contexte KWP2000/K-Line

Les essais réalisés dans le cadre de ce travail ont confirmé que le chemin de diagnostic retenu pour ce véhicule utilise KWP2000 avec une initialisation de type *FastInit* sur ligne K. Contrairement au cas CAN/UDS, l'ouverture d'une session dépend d'une séquence d'initialisation adaptée au calculateur ciblé. Les captures et les scripts de rejeu ont permis d'identifier la séquence suivante :

```
ATWS
ATE0
ATL0
ATS0
ATAL
STPTO 500
STP 25
ATSW 96
ATTA F1
ATWM <en-tete>3E
ATSH <en-tete>
10 C0
```

La réponse positive `50` ou `50C0`, selon le calculateur, confirme l'ouverture de la session. Pour l'injection, l'en-tête utilisé est `817AF1`. Les en-têtes `8126F1`, `812CF1` et `8101F1` ont respectivement permis d'interroger l'UCH, l'airbag et l'ABS. Cette initialisation s'est révélée plus sensible au choix du port, à l'état de l'adaptateur et au changement de session que la communication CAN/UDS étudiée précédemment.

4.7.3 Calculateurs répondants et adresses identifiées

Le sondage K-Line a permis d'identifier quatre calculateurs répondants. Le tableau 4.4 synthétise les résultats confirmés par les journaux expérimentaux.

Tableau 4.4 – Calculateurs répondants identifiés sur la Dacia Duster 2012

ECU / fonction	Adresse cible	Type de réponse obtenu	Preuve / confiance	Rôle dans l'étude VEYA
Injection / moteur	`0x7A`	Session ouverte ; identification et réponses `61xx`	Confirmé par les scripts dédiés	Étude des identifiants locaux, des données moteur et des limites DTC
UCH / contrôleur habitacle	`0x26`	Session ouverte ; identification et réponses `61xx`	Confirmé par sondage XML-guidé	Exploration des informations et défauts mémorisés de l'UCH
Airbag / SRS	`0x2C`	Session ouverte ; réponses `61xx`	Communication confirmée ; interprétation partielle	Vérification de l'accès à un calculateur de sécurité en lecture seule
ABS / VDC	`0x01`	Session ouverte ; identification et enregistrements de défauts	Communication confirmée ; interprétation partielle	Exploration des enregistrements ABS et de leur contexte

4.7.4 Requêtes d'identification et observations `21xx`

Les échanges utiles ne reposent pas uniquement sur les services DTC génériques. Ils utilisent des identifiants locaux constructeur sous la forme `21xx`, avec des réponses positives `61xx`. L'analyse des fichiers XML et des journaux a recensé vingt-neuf identifiants moteur. Lors du sondage XML-guidé documenté, vingt requêtes moteur en lecture seule ont reçu une réponse positive.

Pour l'injection, la requête `2180` fournit des informations d'identification. Les requêtes `21A0`, `21A1`, `21AC`, `21AD` et `21AF` ont été utilisées pour sonder les blocs liés au diagnostic ; `21A9` a également été observée dans les captures RenoLink pour des paramètres moteur. Dans le sondage récent, `21AD` a retourné un contenu non nul, tandis que `21A0`, `21A1`, `21AC` et `21AF` étaient nuls ou majoritairement nuls.

Pour les autres calculateurs, les scripts ont notamment exploité `2194` sur l'UCH, `21C2` sur l'airbag, ainsi que `2180` et les blocs `2191` à `2196` sur l'ABS. Ces réponses confirment l'accès aux données locales, mais elles ne constituent pas encore un décodage exhaustif. Une correction du parseur est notamment nécessaire pour distinguer correctement les charges utiles déjà dépourvues d'en-tête physique des trames qui contiennent encore l'enveloppe KWP.

4.7.5 Évolution de la lecture DTC et limites initiales

La lecture des DTC s'est révélée plus complexe que sur la Symbol CAN/UDS. Sur l'injection, les services testés `17`, `18` et `19` ont retourné des réponses négatives indiquant que le service demandé n'était pas supporté dans la session active. Les requêtes OBD génériques ont fourni une indication compatible avec `P0301`, mais cette correspondance doit encore être comparée au diagnostic de référence avant d'être présentée comme confirmée.

Les blocs locaux ont apporté des informations complémentaires, mais leur interprétation reste partielle. Sur l'UCH, `2194` a fait apparaître des défauts mémorisés sans confirmer clairement la correspondance attendue avec le défaut `DF099`. Sur l'airbag, `21C2` a produit une indication correspondant à deux défauts présents selon le fichier XML candidat ; cette lecture doit encore être comparée à l'outil de référence. Sur l'ABS, les blocs lus contiennent des enregistrements de défauts et de contexte, mais pas encore de libellés directement validés.

Une première méthode de décodage, désignée comme décodeur « v3 », a été invalidée expérimentalement : un bit initialement interprété comme un raté du cylindre 1 est resté actif après un effacement vérifié. Cet effacement a été utilisé uniquement comme étape de comparaison expérimentale pour invalider une interprétation incorrecte ; il ne constitue pas une fonction automatique implémentée dans VEYA. Ce constat montre qu'une association entre un bit et un défaut ne peut pas être retenue sur la seule base d'une hypothèse ou d'un libellé XML. Elle doit être corroborée par une comparaison instrumentée, un effacement suivi d'une nouvelle capture ou le déclenchement contrôlé d'un défaut connu.

Ces limites correspondent aux premières campagnes d'essais et ont orienté la correction de la chaîne d'acquisition. Une campagne finale réalisée avec le dispositif VEYA programmé, l'OBDLink SX et le Raspberry Pi a ensuite permis de lire effectivement les DTC présents sur le véhicule d'essai, notamment `P0300` et `P0420`. Ces codes ont ètè transportés par BLE, affichés sur le tableau de bord et transmis au service distant pour la génération du rapport. La difficulté initiale de lecture ne remet donc pas en cause la validation finale obtenue sur ce véhicule ; elle demeure en revanche pertinente pour rappeler que la généralisation à d'autres calculateurs KWP2000 exige une validation empirique propre à chaque contexte.

4.7.6 Recherche des fichiers XML candidats

La base XML DDT2000 a été parcourue afin de classer les fichiers candidats selon la fonction ECU, la famille de projet, l'adresse, les identifiants observés et la compatibilité des requêtes. Le tableau 4.5 présente les fichiers utilisés lors du sondage XML-guidé. Leur sélection facilite l'exploration, mais ne garantit pas à elle seule la validité de chaque champ décodé.

Tableau 4.5 – Fichiers XML candidats utilisés pour le sondage de la Dacia Duster 2012

ECU	**Fichier XML candidat utilisé**	**Portée de l'exploitation**
Injection / moteur	`SIRIUS32_EMS3130_Soft_REB_460_20100610T153641.xml`	Candidat le mieux classé ; identification et sondage des requêtes locales
UCH	`UCH_transv_7_1_20130122T095956.xml`	Sondage des requêtes UCH et exploration des défauts mémorisés
Airbag / SRS	`AIRBAG_TEMIC_X90_bis_20110120T125412.xml`	Sondage en lecture seule ; interprétation à confirmer par comparaison
ABS / VDC	`ABS_H79_v4.7_20130318T142301.xml`	Lecture des blocs d'identification et des enregistrements de contexte

4.7.7 Bilan de validation et impact sur VEYA

L'essai sur Dacia Duster 2012 a validé un chemin de communication distinct de la chaîne CAN/UDS de la Symbol. L'initialisation KWP2000 de type *FastInit*, l'ouverture de session, l'acquisition des blocs de données vivantes et la lecture des DTC ont ètè réalisées sur le véhicule d'essai au moyen du dispositif VEYA et de l'OBDLink SX commandé par l'ESP32-S3.

Les paramètres de télémétrie exploités par le tableau de bord ont ètè vérifiés en conditions réelles : régime moteur, vitesse, températures, charge, position du papillon,

tension d'alimentation et niveau de carburant. Les DTC effectivement lus sur le véhicule ont ètè transmis jusqu'au Raspberry Pi puis au serveur. Cette campagne confirme donc la faisabilité du chemin KWP2000/K-Line dans la chaîne complète de VEYA.

Cette validation concerne le véhicule et les contextes ECU testés. La généralisation automatique à l'ensemble des calculateurs Renault/Dacia reste conditionnée par la disponibilité des identifiants constructeur et par une validation empirique des décodages. Les résultats ont par ailleurs justifié l'assemblage, sur la même carte, d'une extension K-Line fondée sur le L9637D, dont la validation directe reste une ètape distincte.

4.8 Intégration de la chaîne locale VEYA

4.8.1 Objectif de l'intégration locale

Après l'exploration des protocoles et des outils de diagnostic, l'objectif de cette phase était de relier les blocs fonctionnels de VEYA dans une chaîne opérationnelle. L'intégration associe le dispositif d'acquisition ESP32-S3, la liaison BLE, les services locaux exécutés sur le Raspberry Pi 5 et le tableau de bord Qt6/QML. Elle constitue le lien pratique entre l'acquisition embarquée et l'interface utilisateur.

La couche distante de rapport et de session live est raccordée à cette chaîne locale. Elle reste externe au périmètre embarqué central, mais son fonctionnement de bout en bout a également été vérifié pendant l'intégration.

4.8.2 Validation de la liaison BLE ESP32-S3 / Raspberry Pi

L'intégration de la liaison sans fil a d'abord ètè conduite avec des données simulées èmises par l'ESP32-S3. Cette ètape a permis de valider le contrat applicatif indépendamment de l'acquisition véhicule : le dispositif transmet des messages JSON structurés par notifications BLE et reçoit les commandes envoyées dans le sens inverse. Les fragments reçus sont réassemblés avant le traitement des messages complets.

La validation finale a ensuite ètè réalisée avec des données réelles issues du véhicule. L'ESP32-S3 a acquis les paramètres vivants et les DTC par l'intermédiaire de l'OBDLink SX, puis les a transmis au Raspberry Pi par BLE. Les essais ont confirmé la diffusion continue de la télémétrie et la circulation bidirectionnelle de la commande `query_dtc` et de sa réponse `dtc`, sans interrompre le flux de données vivantes.

Les DTC `P0300` et `P0420`, effectivement présents pendant la campagne finale, ont ètè reçus par le Raspberry Pi et affichés dans l'interface. Cette preuve distingue clairement la phase initiale de test du contrat de messages de la validation finale sur véhicule.

4.8.3 Organisation logicielle sur Raspberry Pi

Le Raspberry Pi 5 assure le relais entre le dispositif d'acquisition et l'interface graphique. Le processus *bridge* gère la liaison BLE, reconstruit les messages reçus et les remet au *core*. Ce dernier centralise le routage, la sélection de la source active et la

circulation des commandes. Le service *ws_server* diffuse ensuite les données sur le WebSocket local consommé par le tableau de bord.

Dans la couche Qt6/QML, le fournisseur de données *VehicleDataProvider* expose les valeurs reçues aux différents écrans. Cette séparation entre transport BLE, logique centrale, diffusion WebSocket et présentation a permis d'intégrer le dispositif ESP32-S3 sans modifier la responsabilité de chaque couche.

La figure 4.4 synthétise le cheminement des données et le raccordement de la couche distante.

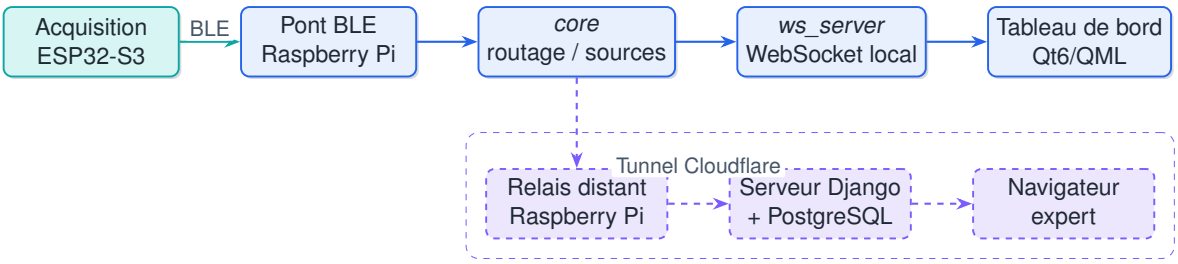


FIGURE 4.4 – Intégration de la chaîne locale VEYA et raccordement au service distant de diagnostic.

4.8.4 Intégration avec le tableau de bord

La télémétrie reçue depuis le dispositif est affichée en temps réel sur le tableau de bord. Les paramètres visualisés comprennent le régime moteur, la vitesse, la température du liquide de refroidissement, la position du papillon, la charge moteur, la tension batterie, le niveau de carburant et la température d'admission. Tous ces champs ont été validés sur le véhicule d'essai par comparaison avec son comportement observé.

La machine d'états du *core* a été vérifiée dans les deux sens. Lorsque le dispositif ESP32-S3 se connecte, la source réelle issue du véhicule remplace la source de démonstration. Lorsque le dispositif se déconnecte, le système revient automatiquement aux données simulées afin de maintenir une interface exploitable.

Le tableau de bord regroupe le profil conducteur, destiné à la visualisation de la télémétrie et des avertissements, ainsi que le profil diagnostic. Il intègre également l'appairage Bluetooth depuis l'écran tactile, dont le cycle appairage, désappairage et réappairage a été vérifié. La configuration Wi-Fi et la gestion des profils conducteur et véhicule sont accessibles depuis l'interface.

4.8.5 Flux des commandes de diagnostic

Le chemin des commandes reprend l'organisation modulaire définie au chapitre précédent. Une requête issue du tableau de bord est remise au *core*, relayée par le pont BLE, puis transmise à l'ESP32-S3. Le dispositif interroge le véhicule et la réponse revient par la même chaîne avant d'être affichée dans l'interface.

La commande `query_dtc` a été validée de bout en bout avec une lecture réelle des codes présents sur le véhicule. L'aller-retour complet entre l'interface, le *core*, le pont BLE, le micrologiciel ESP32-S3, l'OBDLink SX et le calculateur a été exécuté sans suspendre l'affichage continu de la télémétrie.

La commande `clear_dtc` demeure encadrée comme une opération contrôlée. Son message de résultat est pris en charge par le protocole applicatif, sans que ce chapitre ne revendique un effacement automatique sur véhicule dans le flux normal de VEYA.

4.8.6 Liaison avec le service distant

La chaîne locale expose les données nécessaires au service distant de diagnostic. L'envoi des DTC réels vers le serveur Django associé à une base PostgreSQL permet d'obtenir un rapport structuré comprenant la signification des codes, leurs causes probables, les symptômes associés et les réparations recommandées.

La chaîne complète a ètè validée de bout en bout sur matériel et véhicule réels. Depuis le tableau de bord Raspberry Pi, l'utilisateur a déclenché une lecture des DTC ; la requête a ètè transmise par BLE à l'ESP32-S3, exécutée sur le véhicule par l'OBDLink SX, puis renvoyée au Raspberry Pi et au serveur. Le rapport produit a ensuite ètè restitué sur le tableau de bord.

La session live a ègalement ètè confirmée depuis le navigateur de l'expert. Celui-ci a observé la télémétrie réelle en temps réel, déclenché une lecture distante des DTC, reçu le résultat et consulté le détail des codes. Le chemin ESP32-S3–BLE–Raspberry Pi–relais–serveur Django–navigateur expert est donc validé dans les deux sens. Ce service reste une couche distante externe au système embarqué local.

4.8.7 Bilan d'intégration

Cette phase a validé la chaîne fonctionnelle complète de VEYA. Les données vivantes et les DTC sont acquis sur le véhicule par l'ESP32-S3 et l'OBDLink SX, transportés par BLE vers le Raspberry Pi, routés par les services locaux, affichés dans le tableau de bord, puis transmis au serveur pour l'interprétation et la génération du rapport. La même infrastructure permet à l'expert distant d'observer les données et de déclencher une lecture diagnostique en temps réel.

L'intégration a aussi confirmé le basculement automatique entre source de démonstration et source réelle, la communication bidirectionnelle et la continuité de l'affichage pendant les requêtes. L'organisation en couches reste ainsi compatible avec le fonctionnement global effectivement validé.

4.9 Réalisation du prototype matériel retenu

4.9.1 Objectif du prototype matériel

Le prototype matériel retenu vise à fournir à VEYA une interface d'acquisition dédiée, placée à proximité du connecteur OBD-II du véhicule. Il doit réduire la dépendance envers les interfaces commerciales utilisées pendant les phases exploratoires et exposer au Raspberry Pi un contrat de communication maîtrisé.

L'architecture repose sur un ESP32-S3 assurant le traitement embarqué et la liaison BLE avec le tableau de bord. La réalisation a été conduite progressivement : validation du firmware et du transport sans fil avec des données contrôlées, raccordement à la chaîne locale, puis préparation des interfaces destinées à l'acquisition véhicule. Le prototype OBD est maintenant physiquement assemblé et fonctionnel.

4.9.2 Composition matérielle du dispositif

Le microcontrôleur ESP32-S3 constitue le bloc principal du prototype. Il est associé à une voie CAN fondée sur un module MCP2515 avec transcepteur CAN intégré ou associé. Le raccordement au véhicule est assuré par une entrée DB9 destinée au câble OBD2–DB9. Le module MCP2515, son interface SPI et le raccordement CAN sont entièrement assemblés. Cette voie n'a toutefois pas encore èté testée par le micrologiciel sur véhicule.

Une entrée USB femelle destinée au câble OBDLink SX constitue la voie d'acquisition actuellement validée. Les essais ont confirmé l'énumération USB, l'ouverture de l'interface FTDI/VCP, l'initialisation du protocole du véhicule, la lecture des données vivantes et la lecture des DTC. Cette voie est directement pilotée par l'ESP32-S3 puis raccordée au Raspberry Pi par BLE.

Une interface K-Line fondée sur le circuit L9637D est montée sur la même carte prototype. La ligne K issue de la broche 7 du connecteur OBD-II est acheminée par l'adaptation DB9, et l'ensemble des connexions du L9637D est assemblé. Cette extension matérielle n'a pas encore èté validée sur véhicule par un micrologiciel dédié.

Tableau 4.6 – Blocs du prototype matériel et niveau de validation documenté.

Bloc du prototype	**Composant utilisé**	**Rôle**	**Statut**
Contrôle embarqué	ESP32-S3	Gestion de l'interface USB hôte, acquisition et traitement des messages, communication BLE, sélection des modes et commande des voyants.	Assemblé, programmé et validé.
Interface CAN	MCP2515 avec transcepteur CAN	Accès direct au bus CAN du véhicule à travers l'interface DB9.	Matériel assemblé ; validation par micrologiciel et essai sur véhicule à réaliser.
Raccordement véhicule	Adaptation OBD-II / DB9	Acheminement de l'alimentation ainsi que des lignes CAN et K-Line entre le véhicule et le dispositif.	Assemblé et vérifié.
Chemin USB validé	Entrée USB femelle et OBDLink SX	Acquisition des données en temps réel, lecture des codes défauts et communication avec les calculateurs du véhicule.	Validé sur véhicule.
Alimentation	Convertisseur abaisseur XL4015	Conversion de la tension fournie par la prise OBD-II vers un rail régulé de 5 V.	Rail 5 V validé pour une tension d'entrée comprise entre 12 V et 32 V.
Sélection et signalisation	Commutateur de mode et quatre voyants LED	GPIO4 : sélection du mode. LED rouge : présence du rail 5 V. LED jaune sur GPIO5 : mode DB9/MCP2515. LED bleue sur GPIO6 : état de la liaison BLE. LED verte sur GPIO7 : mode USB/OBDLink SX.	Assemblé, programmé et validé.
Interface K-Line	Transcepteur L9637D	Adaptation physique de la ligne K-Line destinée aux communications KWP2000.	Assemblé sur la carte et raccordé à la broche 7 du connecteur DB9 ; essai sur véhicule à réaliser.

La figure 4.5 documente l'assemblage réel de la carte prototype.

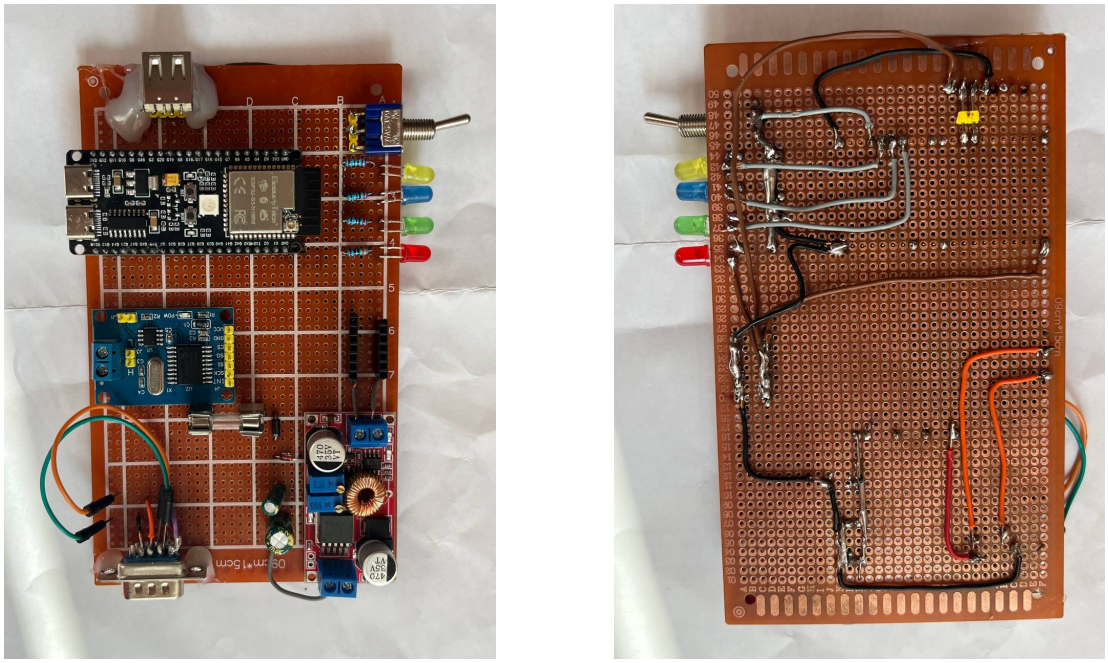


FIGURE 4.5 – Prototype assemblé du dispositif d'acquisition OBD VEYA : vue avant et vue arrière.

Le schéma de câblage du dispositif est présenté à la figure 3.9.

4.9.3 Réalisation de l'alimentation et des protections

L'alimentation du dispositif provient du +12 V disponible sur le connecteur DB9. Le chemin d'alimentation traverse un fusible de 1 A, une diode série standard non Schottky, puis un noeud de protection et de filtrage avant l'entrée du convertisseur abaisseur XL4015. Sur ce noeud, une diode Zener montée en inverse est raccordée à la masse comme branche d'écrêtage, avec deux condensateurs de filtrage de 470 µF et 100 µF vers la masse. La masse du DB9 est raccordée directement à la masse d'entrée du XL4015 et à la masse commune du prototype.

La sortie du XL4015 fournit le rail 5 V régulé. Ce rail alimente l'ESP32-S3, le module MCP2515, l'entrée USB femelle utilisée avec l'OBDLink SX et la LED rouge d'indication 5 V. Les mesures réalisées sur le prototype assemblé ont confirmé que la sortie reste à 5 V pour des tensions d'entrée comprises entre 12 V et 32 V.

4.9.4 Sélection de mode et indicateurs

Un commutateur raccordé au GPIO4 de l'ESP32-S3 est intégré au prototype. Un niveau bas sélectionne le mode DB9/MCP2515 et un niveau haut active le mode USB/OBDLink SX. Le changement de voie est pris en charge par le micrologiciel sans nécessiter une nouvelle programmation.

Une LED rouge raccordée au rail 5 V indique la présence de l'alimentation. La LED jaune sur GPIO5 signale le mode DB9/MCP2515, la LED bleue sur GPIO6 indique l'état de la liaison BLE et la LED verte sur GPIO7 signale le mode USB/OBDLink SX. Le comportement des quatre indicateurs a ète validé sur le prototype assemblé.

4.9.5 Développement du firmware ESP32-S3

Le micrologiciel ESP32-S3 a d'abord ètè validé avec une télémétrie contrôlée afin de stabiliser le contrat applicatif et la communication BLE. Il a ensuite ètè complété par la pile USB hôte, la prise en charge de l'interface FTDI/VCP de l'OBDLink SX, l'initialisation du protocole véhicule, la lecture des données vivantes et des DTC, ainsi que la gestion du commutateur et des voyants.

Le micrologiciel final assure simultanément l'acquisition, le formatage JSON et la communication bidirectionnelle avec le Raspberry Pi. Il reçoit les commandes diagnostiques, exécute les lectures sur le véhicule et transmet les résultats sans interrompre le flux de télémétrie. La sérialisation des messages empêche l'entrelacement des fragments BLE lorsque plusieurs types de trames sont èmis successivement.

La branche matérielle MCP2515 est assemblée, mais sa prise en charge n'a pas encore ètè soumise à une validation sur véhicule. De même, le L9637D est monté et raccordé, tandis que son micrologiciel K-Line direct reste à tester. Ces deux points n'affectent pas le fonctionnement validé de la voie USB/OBDLink SX actuellement utilisée.

4.9.6 État de validation du prototype

Le dispositif OBD VEYA est entièrement assemblé, programmé et fonctionnel. La voie USB hôte avec l'OBDLink SX a ètè validée sur véhicule pour l'acquisition de l'ensemble des paramètres vivants utilisés par le tableau de bord et pour la lecture des DTC. La communication BLE, le changement de mode, les indicateurs et l'intégration avec le Raspberry Pi ont ègalement ètè confirmés.

Le rail d'alimentation 5 V a ètè vérifié pour une entrée comprise entre 12 V et 32 V. La voie CAN personnalisée MCP2515 et l'extension K-Line L9637D sont matériellement assemblées, mais leurs essais directs par micrologiciel sur véhicule restent à conduire. Cette distinction permet de séparer clairement le niveau de réalisation matérielle du niveau de validation expérimentale de chaque voie.

4.9.7 Intégration mécanique et boîtier

Le dispositif est destiné à être regroupé dans un boîtier spécifique imprimé en 3D en PLA permettant de maintenir les cartes et de conserver l'accès aux connecteurs utiles. Le conditionnement tient compte du raccordement OBD-II / DB9, de l'alimentation et des interfaces réservées.

Le dimensionnement mécanique du boîtier OBD est établi autour de la carte prototype, dont l'encombrement est d'environ 9 cm × 15 cm. L'empreinte retenue du boîtier est de 9,5 cm × 15,5 cm, avec une faible marge d'assemblage pour le passage des connecteurs et des câbles ainsi que la fixation interne des cartes. La hauteur du boîtier est comprise entre 2 cm et 3 cm.

Le boîtier imprimé du dispositif est présenté à la figure 3.10.

4.10 Validation globale et discussion

4.10.1 Synthèse de la validation globale

La mise en œuvre expérimentale de VEYA a progressé depuis les premiers essais avec des interfaces commerciales jusqu'à la validation de la chaîne complète. Les tests sur Renault Symbol 2019 ont validé la communication CAN/UDS, l'adressage de plusieurs calculateurs, l'extraction de DTC, la lecture du kilométrage et l'exploitation des captures PCAP avec enrichissement DDT2000.

Les essais sur Dacia Duster 2012 ont complété ces résultats par un contexte KWP2000/K-Line. Après les phases d'identification et de correction des méthodes de décodage, la validation finale a permet l'initialisation de la session, la lecture des données vivantes et la lecture des DTC sur le véhicule testé.

Le chemin embarqué final repose sur l'ESP32-S3 en USB hôte avec l'OBDLink SX, puis sur une liaison BLE vers le Raspberry Pi. Tous les paramètres utilisés par le tableau de bord ont ètè validés en conditions réelles. Les DTC lus sur le véhicule ont traversé la chaîne locale, ont ètè envoyés au serveur Django/PostgreSQL et ont permis de produire un rapport structuré. L'interface expert a affiché de son côté, la télémétrie réelle, déclenché une lecture des DTC et recevoir leur détail. La validation de bout en bout couvre ainsi l'acquisition, le transport, le traitement, l'affichage local, le rapport et la session distante.

Le dispositif matériel est assemblé et le micrologiciel principal est opérationnel. La voie DB9/MCP2515 et l'extension L9637D sont assemblées, mais restent à tester directement par leur micrologiciel sur véhicule.

4.10.2 Niveaux de validation obtenus

Le tableau 4.7 distingue les fonctions validées des deux voies matérielles restant à tester séparément.

Bloc du système	Niveau de validation	Preuve ou résultat	Travail restant
Exploration ELM327 Bluetooth	Validé sur véhicule	Données OBD-II standard explorées sur plusieurs véhicules et limites des clones identifiées	Conserver ces interfaces comme référence comparative
CAN/UDS sur Renault Symbol 2019	Validé sur véhicule	Communication multi-ECU, lecture DTC, kilométrage et analyse PCAP	Étendre progressivement la couverture constructeur
KWP2000/K-Line sur Dacia Duster 2012	Validé sur le véhicule testé	Ouverture de session, données vivantes et DTC réels acquis	Généraliser le décodage à d'autres calculateurs et véhicules
ESP32-S3 / OBDLink SX USB Host	Validé sur véhicule	Énumération FTDI/VCP, commandes AT/ST, acquisition live et lecture DTC	Consolider par des campagnes sur d'autres modèles
BLE ESP32-S3 vers Raspberry Pi	Validé de bout en bout	Télémétrie réelle, commandes et réponses diagnostiques transportées	Optimiser la supervision et la reprise après perte de liaison
Tableau de bord Raspberry Pi	Validé sur données réelles	Affichage de tous les paramètres, DTC, appairage et basculement de source	Poursuivre les améliorations ergonomiques
Rapport et session expert distante	Validé de bout en bout	Rapport issu de DTC réels ; télémétrie et lecture DTC depuis le navigateur	Renforcer l'authentification et les contrôles d'accès
Prototype, alimentation, modes et voyants	Assemblé, programmé et validé	Rail 5 V validé, sélection GPIO4 et quatre indicateurs fonctionnels	Poursuivre les essais de durée et d'environnement
Voie CAN personnalisée MCP2515	Matériel assemblé	Module, SPI et raccordement DB9 achevés	Valider le micrologiciel et la communication CAN sur véhicule
Extension K-Line L9637D	Matériel assemblé	Circuit monté sur la même carte et raccordement K-Line achevé	Valider le micrologiciel et les essais directs sur véhicule

Tableau 4.7 – Synthèse des niveaux de validation obtenus pour les blocs de VEYA.

4.10.3 Discussion des choix expérimentaux

Le passage des clones ELM327 Bluetooth à une interface USB plus maîtrisée n'a pas constitué un simple remplacement de matériel. cela nous a permis de mettre

en place une méthode fondée sur l'apprentissage par capture, la réémission dans des conditions contrôlées, la conservation des réponses brutes et la comparaison des interprétations. Cette démarche est particulièrement importante lorsque l'accès aux données constructeur dépasse le cadre des services OBD-II standard.

Les essais sur Renault Symbol 2019 et Dacia Duster 2012 sont complémentaires. La Symbol a fourni un cas principal de validation CAN/UDS avec lecture de DTC, kilométrage et analyse PCAP. Le Duster a montré qu'un système multi-véhicules doit également prendre en compte les contextes KWP2000 sur ligne K et adapter ses méthodes d'identification. Le Duster a, quant à lui, permis de valider sur véhicule un chemin KWP2000 sur ligne K via l'OBDLink SX piloté par l'ESP32-S3 : initialisation FastInit, ouverture de session, acquisition des données vivantes, lecture des DTC et transmission des résultats jusqu'au tableau de bord et au service distant.

La validation de la chaîne locale et de son raccordement distant montre la faisabilité de l'architecture modulaire retenue. L'acquisition, le transport BLE, le routage central, la diffusion WebSocket, la présentation Qt6/QML et le service distant restent séparés par des interfaces explicites. La couche de rapport et de session live ne remplace pas le diagnostic local : elle l'étend par un service logiciel externe déjà intégré.

La validation du chemin ESP32-S3–OBDLink SX en USB hôte confirme le rôle central de l'ESP32-S3 dans l'acquisition. Il ne constitue plus seulement un relais BLE : il pilote l'interface USB, initialise la session de diagnostic, acquiert la télémétrie et les DTC, puis les transporte vers le Raspberry Pi. La voie commerciale validée sert ainsi de base fonctionnelle complète, tandis que les interfaces MCP2515 et L9637D assemblées pourront être validées ultérieurement comme voies d'acquisition directes.

4.10.4 Limites et perspectives

Les principales validations restantes concernent désormais les deux voies directes assemblées sur le prototype. La première est la voie CAN personnalisée fondée sur le MCP2515, dont le montage, le raccordement SPI et l'adaptation DB9 sont achevés, mais dont le micrologiciel n'a pas encore été validé sur véhicule. La seconde est l'extension K-Line fondée sur le L9637D, montée sur la même carte et raccordée à la ligne K, mais non encore testée directement.

La voie USB/OBDLink SX et le pipeline logiciel complet sont quant à eux fonctionnels. Les perspectives portent donc sur l'extension de la couverture à d'autres marques et calculateurs, l'enrichissement continu de la base de diagnostic et la multiplication des campagnes de validation. Les données constructeur et les fichiers XML DDT2000 doivent continuer à être associés à la conservation des valeurs brutes et à une validation empirique.

Enfin, la session distante validée devra être renforcée par des mesures de sécurité adaptées à un déploiement réel : authentification, contrôle des droits, traçabilité des commandes et chiffrement des échanges. Ces améliorations ne remettent pas en cause la preuve de concept obtenue ; elles préparent son passage vers une solution exploitable à plus grande èchelle.

4.11 Conclusion

Ce chapitre a présenté la mise en œuvre progressive puis la validation expérimentale de VEYA. Les premières campagnes ont permis de caractériser les limites des adaptateurs ELM327 génériques, d'étudier les contextes CAN/UDS et KWP2000 et de construire une méthode fondée sur la conservation des trames brutes et la validation empirique.

Les essais finaux ont confirmé le fonctionnement de la chaîne complète sur véhicule. L'ESP32-S3 pilote l'OBDLink SX en USB hôte, acquiert les données vivantes et les DTC réels, puis les transmet par BLE au Raspberry Pi. Le tableau de bord affiche les paramètres validés, déclenche la lecture des DTC et reçoit le rapport produit par le serveur Django/PostgreSQL. Depuis son navigateur, l'expert observe la télémétrie, lance une requête diagnostique et consulte les résultats. L'ensemble du pipeline, du port OBD jusqu'aux interfaces locale et distante, est ainsi validé de bout en bout.

Le dispositif OBD est entièrement assemblé, programmé et fonctionnel. La sélection de mode, les indicateurs et l'alimentation ont ètè confirmés. La voie CAN personnalisée MCP2515 et l'extension K-Line L9637D sont matériellement achevées, mais leurs essais directs par micrologiciel sur véhicule constituent les principales validations restant à mener.

Ces résultats démontrent la faisabilité technique de VEYA et fournissent une base cohérente pour la conclusion générale, les perspectives d'industrialisation et la valorisation du projet dans le cadre du dispositif Diplôme/Startup.

Conclusion générale

Ce travail a porté sur la conception et la réalisation de **VEYA** (*Vehicle Eye Analysis*), un système embarqué destiné à l'acquisition, au traitement et à la transmission des données de diagnostic automobile accessibles depuis la prise OBD-II. L'objectif principal était de proposer une solution capable d'extraire de manière fiable les informations issues des calculateurs du véhicule, de les afficher localement et de les rendre disponibles à distance pour une exploitation plus avancée.

Dans un premier temps, l'étude théorique a permis de présenter les principaux éléments liés au diagnostic automobile, notamment le bus CAN, le protocole OBD-II, les services UDS, le protocole KWP2000 ainsi que le mécanisme de transport ISO-TP. Cette analyse a constitué une base essentielle pour comprendre l'organisation des échanges entre les calculateurs électroniques et les contraintes associées à l'acquisition des données embarquées.

La solution développée repose sur une architecture composée d'un dispositif embarqué basé sur un ESP32-S3, d'une interface de communication avec le véhicule, d'un tableau de bord exécuté sur Raspberry Pi et d'un serveur distant développé avec Django. L'ESP32-S3 assure la gestion de l'interface USB hôte, la communication avec l'adaptateur OBDLink SX, le traitement des commandes de diagnostic et la transmission des données au tableau de bord à travers une liaison Bluetooth Low Energy. Le Raspberry Pi assure ensuite l'affichage local des paramètres du véhicule, la gestion des demandes de diagnostic et l'échange de données avec le serveur distant.

La mise en œuvre du serveur a permis de centraliser les informations de diagnostic, de conserver l'historique des sessions, d'interpréter les codes défauts et de générer des rapports structurés. Le système offre également la possibilité à un expert distant d'accéder aux données en temps réel et d'envoyer des demandes de diagnostic à travers une interface dédiée. Cette architecture établit ainsi une continuité entre le véhicule, le dispositif embarqué, l'utilisateur local et l'expert distant.

Les essais réalisés ont permis de valider le fonctionnement de la chaîne principale du système. L'interface USB hôte de l'ESP32-S3 a été utilisée avec l'OBDLink SX afin d'établir une communication avec le véhicule. Les données en temps réel, le kilométrage ainsi que les codes défauts ont été acquis puis transmis au tableau de bord. Le fonctionnement de la communication BLE, des voyants d'état, du commutateur de sélection de mode, de l'interface graphique et de la communication avec le serveur a également été vérifié.

La chaîne complète allant de la demande émise depuis le tableau de bord jusqu'à la réception, l'interprétation et la génération du rapport sur le serveur a été validée. Les essais ont également confirmé la possibilité pour un expert d'observer les paramètres du véhicule à distance et de déclencher une demande de lecture des codes défauts.

Ces résultats montrent que la solution développée répond aux principaux objectifs définis au début du projet.

Le prototype matériel intègre également une interface CAN basée sur le contrôleur MCP2515 ainsi qu'une extension K-Line utilisant le transcepteur L9637D. Ces deux interfaces ont été assemblées et intégrées à la carte électronique. Toutefois, leur validation complète par micrologiciel et par essais directs sur véhicule reste à réaliser. Elles constituent néanmoins une base importante pour l'évolution du système vers une architecture plus autonome, capable de communiquer directement avec plusieurs types de réseaux automobiles sans dépendre exclusivement d'un adaptateur externe.

Malgré les résultats obtenus, certaines limites subsistent. Les essais ont été réalisés sur un nombre limité de véhicules et de configurations électroniques. Les différences entre constructeurs, protocoles, vitesses de communication et méthodes d'identification des calculateurs nécessitent encore une phase importante de validation. L'interprétation des données propriétaires dépend également de la disponibilité et de la qualité des bases de données utilisées.

Plusieurs perspectives peuvent être envisagées pour poursuivre le développement de VEYA. Il serait notamment possible de valider complètement les interfaces MCP2515 et L9637D, d'étendre les essais à un plus grand nombre de marques et de modèles, d'améliorer l'identification automatique des calculateurs et d'enrichir la base de données de diagnostic. L'intégration de méthodes d'analyse intelligente pourrait également permettre la détection précoce des anomalies, la comparaison de plusieurs sessions et l'élaboration de recommandations plus précises.

D'autres améliorations pourraient concerner la sécurisation des communications, la gestion de plusieurs véhicules et utilisateurs, l'optimisation de l'interface graphique ainsi que le développement d'un boîtier plus compact et mieux adapté à une utilisation quotidienne. Une évolution vers une carte électronique personnalisée regroupant l'alimentation, les interfaces CAN et K-Line, la communication sans fil et le traitement embarqué permettrait également de réduire le coût, l'encombrement et la dépendance à des modules externes.

En conclusion, ce projet a permis de réaliser un système fonctionnel combinant l'électronique embarquée, les réseaux automobiles, les communications sans fil, le développement logiciel et les technologies Web. Les résultats obtenus démontrent la faisabilité d'une solution de diagnostic automobile connectée, évolutive et accessible à distance. VEYA constitue ainsi une base solide pour le développement futur d'un système plus complet, capable de faciliter l'accès aux données du véhicule et de rapprocher l'utilisateur de l'expertise automobile.

Bibliographie

[1] Long Chen and Zhuo Li. LabVIEW-based CAN bus data acquisition and monitoring system for automobiles. In *2022 IEEE 2nd International Conference on Electronic Technology, Communication and Information (ICETCI)*, pages 1292–1295. IEEE, 2022.

[2] Kyong-Tak Cho and Kang G. Shin. Fingerprinting electronic control units for vehicle intrusion detection. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 911–927, Austin, TX, 2016. USENIX Association.

[3] CSS Electronics. CAN Bus : The Ultimate Guide. Technical guide, 2023. PDF guide, last modified 21 June 2023. Accessed : 2026.

[4] CSS Electronics. CSS Electronics Catalog v4. Product catalog, 2023. Catalog of CLX000, CANedge and CANmod CAN hardware. Accessed : 2026.

[5] ELM Electronics. *ELM327 OBD to RS232 Interpreter – Datasheet*. ELM Electronics Inc., version 2.3 edition, 2017. `https://www.elmelectronics.com/wp-content/uploads/2016/07/ELM327DS.pdf`.

[6] Oliver Hartkopp and python-can contributors. python-can : Controller Area Network interface module for Python. `https://github.com/hardbyte/python-can`, 2024. Accessed : 2025.

[7] Oliver Hartkopp, Urs Thürmann, and Matthias Schilling. SocketCAN – the official CAN API of the Linux kernel. Linux Kernel Documentation, 2024. `https://www.kernel.org/doc/html/latest/networking/can.html`.

[8] Tobias Hoppe, Stefan Kiltz, and Jana Dittmann. Security threats to automotive CAN networks – practical examples and selected short-term countermeasures. *Reliability Engineering & System Safety*, 96(1) :11–25, 2011.

[9] International Organization for Standardization. ISO 11898-1 : Road vehicles – controller area network (CAN) – part 1 : Data link layer and physical signalling. International standard, ISO, 2015.

[10] International Organization for Standardization. ISO 15031-5 : Road vehicles – communication between vehicle and external equipment for emissions-related diagnostics – part 5 : Emissions-related diagnostic services. International standard, ISO, 2015.

[11] International Organization for Standardization. ISO 14230 : Road vehicles – diagnostic communication over K-Line (DoK-Line) – keyword protocol 2000. International standard, ISO, 2016.

[12] International Organization for Standardization. ISO 15765-2 : Road vehicles – diagnostic communication over controller area network (DoCAN) – part 2 : Transport protocol and network layer services. International standard, ISO, 2016.

[13] International Organization for Standardization. ISO 14229-1 : Road vehicles – unified diagnostic services (UDS) – part 1 : Application layer. International standard, ISO, 2020.

[14] Jheng-Syu Jhou, Shi-Huang Chen, Wu-Der Tsay, and Mei-Chiao Lai. The implementation of OBD-II vehicle diagnosis system integrated with cloud computation technology. In *2013 Second International Conference on Robot, Vision and Signal Processing (RVSP)*, pages 9–12. IEEE, 2013.

[15] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, and Stefan Savage. Experimental security analysis of a modern automobile. In *2010 IEEE Symposium on Security and Privacy*, pages 447–462. IEEE, 2010.

[16] Pier-Yves Lessard. python-can-isotp : ISO-TP (ISO 15765-2) implementation in Python. `https://github.com/pylessard/python-can-isotp`, 2024. Accessed : 2025.

[17] Keith McCord. *Automotive Diagnostic Systems : Understanding OBD-I and OBD-II*. CarTech Inc., 2011.

[18] Microchip Technology. *MCP2515 Stand-Alone CAN Controller with SPI Interface – Datasheet*. Microchip Technology Inc., 2019. Document DS20001801J.

[19] Charlie Miller and Chris Valasek. A survey of remote automotive attack surfaces. Black Hat USA, Technical White Paper, 2014. `https://ioactive.com/pdfs/IOActive_Remote_Attack_Surfaces.pdf`.

[20] Nicolas Navet and Françoise Simonot-Lion, editors. *Automotive Embedded Systems Handbook*. Industrial Information Technology. CRC Press, 2008.

[21] Dominique Paret and Hassina Rebaine. *Réseaux de communication pour systèmes embarqués : CAN, CAN FD, LIN, FlexRay, Ethernet*. Technique et ingénierie. Dunod, Paris, 2e edition, 2014.

[22] Tomáš Pawlenka, Jiří Kulhánek, Petr Tomčík, and Rostislav Zapletal. Design of digital CAN based car dashboard unit. In *2019 IEEE International Conference on Electrical Engineering*, pages 1–4. IEEE, 2019.

[23] Konrad Reif, editor. *Automotive Mechatronics : Automotive Networking, Driving Stability Systems, Electronics*. Bosch Professional Automotive Information. Springer Vieweg, 2014.

[24] Robert Bosch GmbH. *CAN Specification Version 2.0*. Robert Bosch GmbH, Stuttgart, Germany, 1991.

[25] Society of Automotive Engineers. SAE J1962 : Diagnostic connector equivalent to ISO/DIS 15031-3. Sae standard, SAE International, 2012.

[26] Society of Automotive Engineers. SAE J1979 : E/e diagnostic test modes. Sae standard, SAE International, 2014.

[27] Wilfried Voss. *A Comprehensible Guide to Controller Area Network*. Copperhill Media, 2nd edition, 2008.

[28] Quanqi Wang, Jian Wang, and Yanyan Wang. Design of vehicle bus data acquisition and fault diagnosis system. In *2011 International Conference on Consumer Electronics, Communications and Networks (CECNet)*, pages 245–248. IEEE, 2011.

[29] Jonathan Wass. rpi-can-logger : CAN bus data logger for Raspberry Pi with PiCAN2. `https://github.com/JonnoFTW/rpi-can-logger`, 2020. Accessed : 2025.

ANNEXE A

Organisation logicielle du tableau de bord VEYA

A.1 Présentation générale

Cette annexe présente l'organisation logicielle du tableau de bord VEYA déployé sur Raspberry Pi. Cette couche assure l'affichage local des données issues du dispositif ESP32-S3, la validation des messages reçus, la circulation des mesures vers le serveur Django et la prise en charge des sessions d'expertise distante.

A.2 Arborescence simplifiée du projet

Les dossiers générés, temporaires ou sans rapport direct avec l'architecture du tableau de bord ne sont pas repris ici.

```
veya/
|-- docs/
|   |-- ARCHITECTURE.md
|   |-- CONTRACT.md
|   |-- STATE_MACHINE.md
|   `-- VEYA_SERVER_CONTRACT_v1.md
|
|-- scripts/
|   |-- setup_bluetooth.sh
|   |-- backup_configs.sh
|   `-- dev_live_session_stub.py
|
|-- services/
|   `-- veya_core/
|       |-- ble_bridge.py
|       |-- contract.py
|       |-- core.py
|       |-- live_session.py
|       |-- tcp_server.py
|       |-- ws_server.py
|       |-- helpers/
|       |   |-- bluetooth.py
|       |   |-- server_client.py
|       |   |-- wifi.py
|       |   |-- load_profile.py
|       |   |-- save_profile.py
|       |   `-- set_transport.py
|       `-- sources/
```

```
|           |-- elm327_source.py
|           `-- mock_source.py
|
|-- ui/
|   |-- CMakeLists.txt
|   |-- src/
|   |   `-- main.cpp
|   `-- qml/
|       |-- Main.qml
|       |-- Home.qml
|       |-- Drive.qml
|       |-- Diagnostic.qml
|       |-- ReportScreen.qml
|       |-- LiveSessionScreen.qml
|       |-- SettingsScreen.qml
|       |-- components/
|       |-- providers/
|       |-- onboarding/
|       `-- assets/
|
|-- start_veya.sh
|-- start_mock.sh
`-- requirements.txt
```

A.3 Couche de services Python

Le dossier `services/veya_core/` regroupe les services locaux du tableau de bord. Le module `ble_bridge.py` assure la liaison BLE avec l'ESP32-S3 et transmet les messages reçus vers le noyau applicatif. Le fichier `core.py` centralise la logique de coordination, tandis que `contract.py` valide la structure des messages, notamment les trames JSON de télémétrie, les réponses de lecture DTC et les commandes applicatives.

Les fichiers `tcp_server.py` et `ws_server.py` exposent les services locaux TCP et WebSocket utilisés par l'interface graphique et par les composants externes. Le module `live_session.py` prend en charge les sessions d'expertise distante, en relayant les événements utiles vers le serveur Django et l'interface de l'expert. Les aides regroupées dans `helpers/` gèrent la configuration Bluetooth, le client serveur, le Wi-Fi, les profils et le choix du transport. Les sources `elm327_source.py` et `mock_source.py` permettent respectivement l'utilisation d'une source OBD compatible et d'une source de simulation.

A.4 Interface graphique Qt 6/QML

Le dossier `ui/` contient l'application graphique Qt 6/QML. Le fichier `src/main.cpp` initialise l'application native et charge les vues QML. Les écrans `Home.qml`, `Drive.qml`, `Diagnostic.qml`, `ReportScreen.qml`, `LiveSessionScreen.qml` et `SettingsScreen.qml` structurent les fonctions visibles du tableau de bord. Les sous-dossiers `components/`, `providers/`, `onboarding/` et `assets/` regroupent les composants réutilisables, les fournisseurs de données, les écrans d'accueil et les ressources graphiques.

A.5 Accès au dépôt du projet

Le script `start_veya.sh` lance la pile normale du tableau de bord sur la plateforme embarquée. Le script `start_mock.sh` démarre une configuration de simulation utile pour les essais sans véhicule. Les dépendances Python nécessaires sont regroupées dans `requirements.txt`.

Le code source du tableau de bord VEYA est disponible dans le dépôt Git du projet :

`https://github.com/Ahmed2002-rak/veya.git`

ANNEXE B

Organisation du micrologiciel ESP32-S3 sous ESP-IDF

B.1 Présentation du projet ESP-IDF

Cette annexe décrit l'organisation du projet ESP-IDF utilisé pour le micrologiciel ESP32-S3. Le programme final intègre la gestion des GPIO, des voyants, du BLE, de l'USB Host, de l'interface OBDLink SX et de l'échange de messages JSON avec le tableau de bord. Un micrologiciel antérieur a également servi à valider la compatibilité BLE avant l'intégration complète.

B.2 Arborescence du micrologiciel

```
obdsx_veya_final/
|-- CMakeLists.txt
|-- dependencies.lock
|-- sdkconfig
|-- sdkconfig.defaults
|-- README_TEST_PHASE_1.md
‘-- main/
    |-- CMakeLists.txt
    |-- idf_component.yml
    |-- veya_duster_obdsx_ble_test.cpp
    ‘-- veya_final_firmware.cpp
```

B.3 Fichiers de compilation et de configuration

Fichier	Rôle
`CMakeLists.txt`	Point d'entrée de la compilation ESP-IDF et déclaration du projet.
`main/CMakeLists.txt`	Déclaration du composant principal et des fichiers sources C++ compilés.
`main/idf_component.yml`	Déclaration des composants ESP-IDF gérés et des dépendances logicielles.
`sdkconfig`	Configuration complète produite pour la cible ESP32-S3.
`sdkconfig.defaults`	Paramètres de configuration conservés comme base reproductible.
`dependencies.lock`	Verrouillage des versions de composants gérés par ESP-IDF.
`README_TEST_PHASE_1.md`	Notes de validation et de mise en route de la première phase d'essai.
`veya_duster_obdsx_ble_test.cpp`	Micrologiciel intermédiaire utilisé pour les essais de compatibilité BLE.
`veya_final_firmware.cpp`	Micrologiciel C++ final intégrant l'acquisition, les commandes et la communication avec le tableau de bord.

B.4 Organisation fonctionnelle du programme principal

```
app_main()
|-- initialisation des GPIO et des voyants
|-- initialisation BLE/NimBLE
|-- gestion du commutateur de mode
|-- initialisation USB Host
|-- ouverture de l'OBDLink SX
|-- configuration KWP2000
|-- acquisition des donnees en temps reel
|-- lecture des DTC
|-- traitement des commandes BLE
‘-- transmission des messages JSON
```

Cette organisation maintient la distinction entre la voie USB/OBDLink SX validée sur véhicule et les voies directes MCP2515 et L9637D assemblées mais encore à valider par micrologiciel et essais directs.

B.5 Dépendances logicielles

Le micrologiciel s'appuie sur FreeRTOS, NimBLE, l'USB Host ESP-IDF, CDC-ACM, VCP, le pilote FTDI VCP et les composants gérés par ESP-IDF. Ces dépendances fournissent les tâches temps réel, la communication BLE, l'énumération USB, l'accès au port série virtuel et l'intégration de l'interface OBDLink SX.