HEH be Sciences et technologies

Avenue V. Maistriau 8a
B-7000 Mons
+32 (0)65 33 81 54
scitech-mons@heh.be

WWW.HEH.BE

Gestion de projet

Projet Voiture 2IRT

Choix technologiques

Année Académique
2024 - 2025

Groupe
5

Membres
Colle Joulian
Deneyer Tom
Kruczynski Mathis
Mauroit Antoine
Staquet Esteban
Vangeebergen Augustin

Table des matières

1 Choix Technologiques Matériels

1.1 Le Raspberry Pi comme Solution Optimale

Le choix du Raspberry Pi 3 Modèle B comme plateforme centrale pour ce projet s'est imposé face aux microcontrôleurs traditionnels comme Arduino pour plusieurs raisons fondamentales :

- **Puissance de calcul** : Contrairement aux microcontrôleurs classiques, le Raspberry Pi intègre un processeur quad-core à 1.2GHz et 1GB de RAM, permettant d'exécuter un système d'exploitation complet et de gérer simultanément :
  - Le traitement des données des capteurs en temps réel
  - Les algorithmes de navigation (relativement) complexes
  - La communication réseau
- **Connectivité intégrée** : Le Raspberry Pi offre nativement :
  - Wi-Fi 802.11n et Bluetooth 4.1 pour les communications sans fil
  - 4 ports USB pour connecter des périphériques additionnels
  - Interface HDMI pour le débogage
- **Multitasking** : La capacité à exécuter plusieurs processus en parallèle est cruciale pour :
  - Gérer simultanément les capteurs et les moteurs
  - Maintenir une connexion SSH active
  - Exécuter un serveur web pour l'interface de contrôle
- **Écosystème logiciel** : L'environnement Linux permet d'utiliser :
  - Python 3 comme langage principal avec toutes ses bibliothèques
  - Des outils professionnels de versioning (Git)
  - Des frameworks de test et d'intégration continue

1.2 Limitations des Microcontrôleurs Traditionnels

Les solutions comme Arduino ou ESP32, bien que performantes pour certaines applications, présentent des limitations majeures pour ce projet :

- **Capacité de traitement insuffisante** pour les algorithmes avancés de détection d'obstacles et d'optimisation de trajectoire
- **Absence de système d'exploitation** rendant complexe :
  - La gestion concurrente des tâches
  - Le débogage avancé
  - Les communications réseau
- **Connectivité limitée** nécessitant des modules additionnels pour certaines fonctionnalités :
  - Le stockage de données
  - Les interfaces utilisateur avancées
- **Espace mémoire restreint** incompatible avec :
  - Le stockage des logs de diagnostic
  - L'exécution de bibliothèques complexes
  - La gestion de protocoles réseau complets

1.3 Comparaison avec d'Autres Solutions

TABLE 1 – Comparatif des plateformes matérielles

Caractéristique	**Raspberry Pi 3**	**Arduino Mega**	**ESP32**
Processeur	Quad-core 1.2GHz	16MHz	Dual-core 240MHz
Mémoire	1GB RAM	8KB RAM	520KB RAM
Système d'exploitation	Linux	Aucun	FreeRTOS
Connectivité	Wi-Fi/BT intégrés	Requiert shields	Wi-Fi/BT intégrés
Langages supportés	Python, C++, ...	C/C++	C/C++, MicroPython
Prix	~50€	~40€	~5-10€

Ce tableau montre clairement l'avantage du Raspberry Pi en termes de rapport performance/prix pour les besoins spécifiques de ce projet. Il est toutefois à noter que l'ESP32 présente un excellent compromis pour des applications plus légères avec sa connectivité sans fil intégrée.

1.4 Proposition d'alternative et son rejet

Bien que, comme mentionné précédemment, l'ESP32 présente certains avantages pour des applications légères, plusieurs facteurs critiques justifient le maintien des Raspberry Pi 3B dans ce projet pédagogique :

- **Compatibilité avec le parc existant** :
  - L'école dispose déjà d'une flotte de Raspberry Pi 3B fonctionnels
  - Tous les accessoires périphériques (écrans, câbles, alimentations) sont prévus pour le format Pi
  - Les supports pédagogiques actuels sont spécifiquement conçus pour Raspberry Pi, sans compter la communauté et la documentation abondante disponible en ligne.
- **Avantages pédagogiques** :
  - Initiation à Linux, système d'exploitation professionnel
  - Environnement de développement complet (IDE, debuggeur)
  - Meilleure visualisation des concepts avec interface graphique
  - Compatibilité avec les outils standards (Git, SSH, VNC)
- **Limitations techniques de l'ESP32** :
  - Impossible d'exécuter simultanément :
    - L'interface utilisateur
    - Le serveur web de contrôle
    - Le logging avancé des données
  - Mémoire insuffisante pour le débogage interactif
  - Connectivité HDMI absente nécessitant des adaptateurs supplémentaires
- **Économie à court terme, coût à long terme** :
  - Coût de migration des développements existants
  - Nécessité de reformer les enseignants et étudiants
  - Achat de nouveaux accessoires (programmateurs, shields)
  - Perte de compatibilité avec les projets antérieurs

- **Benchmarks comparatifs** : Les tests pratiques démontrent des limitations majeures de l'ESP32 pour notre cas d'usage :

TABLE 2 – Comparaison des performances réelles

Tâche	**RPi 3B**	**ESP32**
Traitement d'image (640x480)	15 fps	Impossible
Serveur web + interface	Fluide	Non réalisable
Stockage des logs (1h)	Direct	Nécessite SD card
Débogage temps réel	Possible	Très limité

Conclusion : Bien que techniquement envisageable pour des applications minimalistes, le remplacement des Raspberry Pi par des ESP32 compromettrait la qualité pédagogique et l'expérience d'apprentissage, sans réelle économie à l'échelle du parc existant. Et c'est sans compter l'affichage d'images de la webcam dans le cas où cette fonctionnalité serait implémentée dans les temps.

2 Choix Technologiques Logiciels

2.1 Système d'Exploitation

Le choix s'est porté sur **Raspberry Pi OS** (version GUI, 32bits) pour plusieurs raisons techniques :

- **Compatibilité matérielle** : Support natif de tous les composants du projet
- **Stabilité** : Distribution officielle maintenue par la fondation Raspberry Pi (mieux qu'une distribution obscure IOT pour laquelle certains drivers manqueraient).
- **Outils intégrés** : Inclut par défaut :
  - Gestionnaire de paquets apt
  - Supports Hotspot et WiFi

2.2 Langage de Programmation

Python 3 a été sélectionné comme langage principal pour :

- **Productivité** : Syntaxe claire permettant un développement rapide
- **Polyvalence des paradigmes** :
  - Prise en charge quasi-complète de la programmation orientée objet (classes, héritage, polymorphisme)
  - Possibilité de programmer de manière procédurale pour les scripts simples
  - Éléments de programmation fonctionnelle (fonctions lambda, map, filter)
  - Flexibilité permettant d'adapter le style de programmation à chaque composant du système
- **Bibliothèques spécialisées** :
  - board (récupération des lignes sda/scl par défaut)
  - busio (pour l'utilisation des bus I2C)
  - adafruit_ina219

  - adafruit_pca9685
  - adafruit_tcs34725
  - rpi.gpio
  - unittest
- **Communauté active** : Large base d'utilisateurs et documentation abondante
- **Portabilité** : Fonctionne sur tous les systèmes d'exploitation (ou presque)

2.3 Outils de Développement

L'environnement logiciel comprend :

- **Git - Github** : Pour le versioning et la collaboration en équipe
- **SSH/TightVNC** : Permet le développement à distance
- **UnitTest** : Framework de tests unitaires intégré à Python

L'environnement non-logiciel comprend :

- **Notion** :
  - pour la gestion des taches en temps réel pour le scrumboard
  - visualisation du planning
  - centraliser la documentation
  - gérer les daily scrums
  - reviews
  - retrospectives
  - journal de bord accessible par toute l'équipe
  - gestion des user-stories

2.4 Architecture Logicielle

L'application suit une architecture modulaire :

- **Couche matérielle** : Drivers pour les capteurs et actionneurs
- **Couche contrôle** : Algorithmes de navigation et détection, tests unitaires
- **Couche communication** : Gestion des connexions réseau, SSH, I2C
- **Interface utilisateur** : TUI à travers le SSH

Cette architecture permet une maintenance aisée et une bonne séparation des préoccupations.

2.5 Approche de Test et Validation

Une méthodologie rigoureuse de test a été mise en place pour garantir la fiabilité du système tout en protégeant le matériel :

- **Philosophie de test** :
  - **Tests avant implémentation** : Développement piloté par les tests (TDD) pour les composants critiques
  - **Protection du matériel** : Validation logicielle complète avant tout déploiement sur le Raspberry Pi, utilisation de capteur de protection et de préactionneurs (pont en H)

  - **Isolation des défauts** : Tests unitaires pour identifier précisément l'origine des problèmes
- **Infrastructure de test** :
  - Framework `unittest` de Python comme base
  - Création de **mock objects** pour :
    - Simuler les capteurs (GPIO, I2C) sans risque de court-circuit
    - Émuler des conditions extrêmes (tension trop élevée, température critique)
    - Tester les cas d'erreur sans endommager le matériel
  - Bancs de test virtuels pour valider les algorithmes de contrôle
- **Exemple concret de mock** :

```
import unittest
from unittest.mock import MagicMock, patch
from vehicle import VehicleController

class MockMotor():
        def __init__(self, name):
                self.status = f"Motor {name} initialized"

        def activate(self):
                print("Motor activation signal sent")
                return "Motor activated"

class TestVehicleSystem(unittest.TestCase):

        @patch("vehicle.Motor", new=MockMotor)
        def test_vehicle_control(self):
                controller = VehicleController("main_motor",
                   100)
                result = controller.activate_motor()
                self.assertEqual(result, "Motor activated")
                print("Test passed: Motor control OK")

if __name__ == "__main__":
        unittest.main()
```

- **Avantages de cette approche** :
  - **Sécurité matérielle** : Aucun risque de griller des composants pendant le développement
  - **Reproductibilité** : Conditions de test parfaitement contrôlées
  - **Efficacité** : Tests exécutables sans le matériel physique
  - **Couverture** : Possibilité de simuler des cas rares ou dangereux
- **Validation progressive** :
  1. Tests unitaires avec mocks (100% de couverture)
  2. Tests d'intégration en environnement simulé
  3. Tests système sur banc de validation
  4. Déploiement sur le prototype physique

Cette méthodologie nous permet de développer en toute sécurité des fonctionnalités complexes comme le contrôle PID des moteurs ou la gestion des capteurs I2C, tout en minimisant les risques pour le matériel coûteux. Les mocks reproduisent fidèlement le comportement des composants physiques, y compris leurs temps de réponse et modes de défaillance.