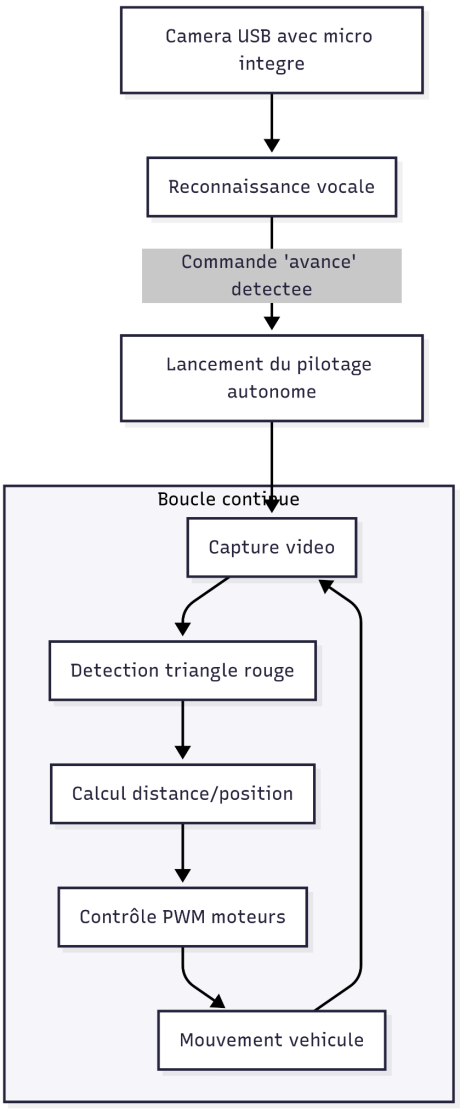
Documentation Technique : Système de Voiture Autonome Raspberry Pi

1. Présentation du système

Système embarqué transformant une voiture radiocommandée classique en véhicule autonome avec :

- Reconnaissance vocale pour déclencher le mode autonome
- Vision par ordinateur pour la navigation
- Contrôle PWM des moteurs/servomoteurs
- Accès SSH pour le contrôle et surveillance à distance

Flux opérationnel :



2. Architecture matérielle

Composant	**Spécifications**	**Rôle**
Raspberry Pi 4	4GB RAM, Wi-Fi intégré (pour connexion ssh)	Cerveau du système, héberge l'algo python
Module PCA9685	Contrôleur PWM I2C 16 canaux	Pilotage moteur/servo de la voiture
Webcam USB	Résolution réduite à 736×414 px	Fourni les images pour la computer vision
Micro USB	Microphone intégré à la caméra	Capture audio pour le lancement avec "avance"
Batterie Externe	5V/3A	Alimentation mobile pour la raspberry
Batterie LiPo	2S (14.8v)	Alimentation de la voiture (moteur/servo)

3. Installation logicielle

Prérequis :

```
sudo apt install python3-pip
pip install pocketsphinx sounddevice opencv-python numpy
adafruit-circuitpython-pca9685
```

Structure des fichiers :

```
/voiture_autonome/
├── auto.py             # Script pilotage auto avec CV
└── launch.py           # Script avecc reconnaissance vocale
pour lancer la conduite auto
```

4. Configuration audio

Sélection du périphérique :

1. Lister les périphériques :

```
import sounddevice as sd
print(sd.query_devices())
```

2. Choisir l'index du micro USB
3. Tester avec `test_micro(device_index)`

Paramètres PocketSphinx :

```
speech = LiveSpeech(
    hmm='/usr/share/pocketsphinx/model/fr-fr-ptm-5.2',

dict='/usr/share/pocketsphinx/model/fr-fr-ptm-5.2/fr.dict',
    keyphrase='avance',
    kws_threshold=1e-20,
    audio_device=MIC_DEVICE_INDEX
)
```

5. Système de vision

Détection du triangle rouge :

1. Conversion HSV : `cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)`
2. Masquage rouge :

```
lower_red1 = [0, 100, 100]
upper_red1 = [10, 255, 255]
lower_red2 = [160, 100, 100]
upper_red2 = [180, 255, 255]
```

3. Détection de contours et filtrage des triangles

4. Estimation de distance :

```
def estimate_distance(w):
    return (5.0 * 240) / w  # largeur du triangle = 5cm, focal_length = 240px
```

6. Contrôle des moteurs

Mapping PWM :

Commande	**Valeur 12-bit**	**Valeur 16-bit**
NEUTRAL	307	4915
FORWARD	410 et +	6553 et +
LEFT	520	8308
RIGHT	105	1677

Algorithme de contrôle :

```
# Throttle proportionnel à la distance du triangle

if distance > 23:  # min_distance
    ratio = (distance - 23) / (275 - 23)  # max_distance=275
    throttle = NEUTRAL + ratio * (FORWARD - NEUTRAL)/2

# Steering avec atténuation à haute vitesse

steering_range = 1.0 - (throttle_ratio * 0.999)
steering = NEUTRAL + (offset / max_offset) * (RIGHT - NEUTRAL)
* steering_range
```

7. Journalisation et débogage

Commandes SSH utiles :

```
# Surveillance ressources cpu ram,...
htop

# Vérification I2C
i2cdetect -y 1

# Test caméra
libcamera-hello

# Lancement du programme
python3 launch.py
#une fois le programme lancé, la raspberry renvoie en direct
dans le terminal les valeurs de throttle et directions
appliquées à la voiture
```

8. Sécurité et gestion d'erreurs

Protections intégrées :

1. Retour au neutre pour le moteur et la direction à la fermeture du programme
2. Retour au neutre pour le moteur et la direction si le triangle n'est plus détecté
3. Timeout vidéo :

```
cap.set(cv2.CAP_PROP_FRAME_TIMEOUT, 1000)
```

4. Clipping PWM :

```
np.clip(raw_steering, min(LEFT, RIGHT), max(LEFT, RIGHT))
```

9. Optimisations possibles

1. Multithreading :

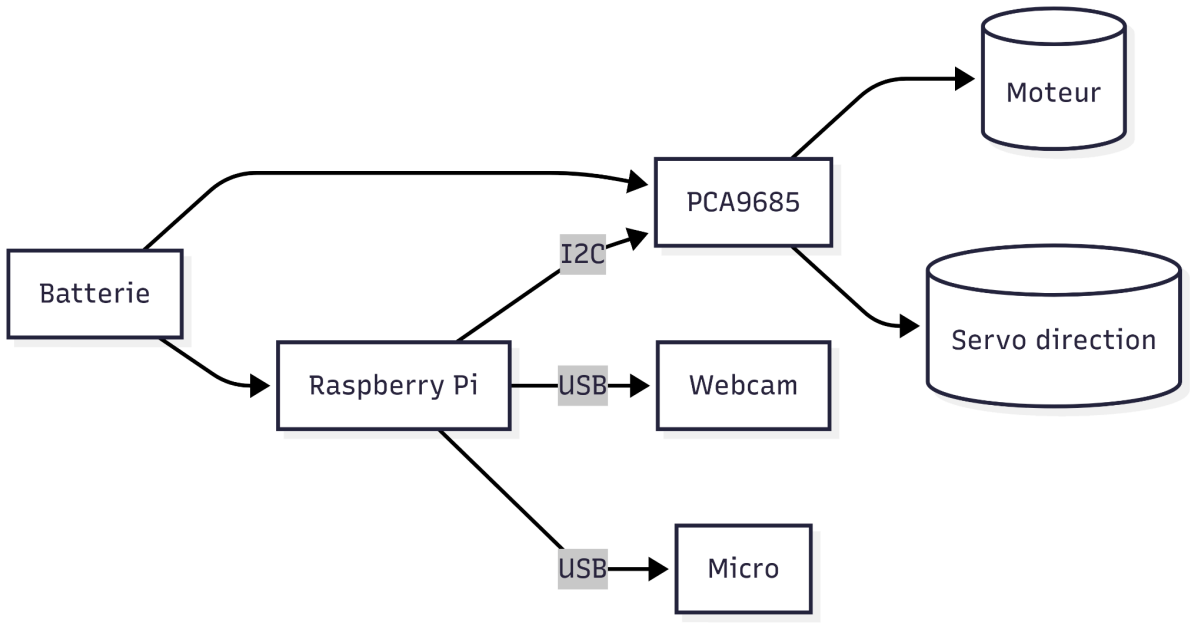
```
from threading import Thread
Thread(target=vision_processing).start()
```

2. Compilation OpenCV avec optimisations NEON
3. Overclocking GPU :

```
sudo nano /boot/config.txt
gpu_freq=500
```

4. Utilisation de Cython pour fonctions critiques

11. Schéma de câblage



12 . Interface Web

1. remote_control2.py — Serveur Python (Flask)

Fonctionnement général

- Lance un serveur HTTP sur localhost:5000.
- Fournit des routes (endpoints) comme /forward, /left, /stop, etc.
- Utilise le module adafruit_pca9685 pour contrôler la direction et la vitesse du véhicule via des signaux PWM.

Configuration matérielle

- Communication I2C pour le contrôle du module PCA9685.
- Fréquence PWM définie à 50 Hz (typique pour servos).
- Deux canaux utilisés :
    - THROTTLE_CHANNEL = 0 → propulsion (avant/arrière)
    - STEERING_CHANNEL = 1 → direction (gauche/droite)

Endpoints disponibles

Route	Action
/forward	Avance pendant 3 secondes à 50%
/backward	Recule pendant 3 secondes à 50%

/left	Tourne les roues vers la gauche
/right	Tourne les roues vers la droite
/stop	Arrête propulsion et direction

Utilisation de threading.Thread pour ne pas bloquer le serveur avec sleep().

2. super_interface_thomas2.html — Interface Web de Contrôle

Commandes manuelles

Bouton	Action JS	Commande envoyée
Avancer	send('forward')	/forward
Gauche	send('left')	/left
Droite	send('right')	/right
Stop	send('stop')	/stop

13 . Améliorations futures

1. Intégration SLAM avec carte de profondeur
2. Réseau neuronal pour détection d'objets plus poussés (autre que triangle rouge)
3. Télémétrie dans l’axe de la caméra avec télémètre laser pour gagner en précision
4. Interface web de contrôle avec retour caméra
5. Implémentation d’un double contrôle avec la télécommande pour pouvoir reprendre le contrôle sur la voiture

Cette documentation couvre l'ensemble du système technique. Pour des implémentations spécifiques, se référer aux commentaires détaillés dans les codes source directement.