Ahoumsy DALARUN

Jules LE PAIH

Bogdan MOUMY EMBOLA

Duong Thien An NGUYEN

GEII 1

Rapport d'études et réalisations

Semestre 4

Course ENS voiture autonome

Complément au rapport de Semestre 3

Professeur référent : M Ruiz

2023 – 2024

INTRODUCTION

Ceci est un rapport écrit par Ahoumsy DALARUN, Jules LE PAIH, Bogdan MOUMY EMBOLA et Duong Thien An NGUYEN, élèves de l'IUT de Cachan. Il agit en tant que complément au rapport rédigé par les élèves de GEII 2. Dans le cadre de séances de mise en situation d'apprentissage évaluées, nous devons réaliser un système de détection de couleur ainsi qu'un système de détection de véhicule. Ces systèmes seront par la suite intégrés à la voiture autonome réalisée par les élèves du secteur GEII 2, participants à la course ENS voiture autonome 2024. Le projet permet de nous initier au traitement d'image, et de nous apprendre le déroulement d'un projet en entreprise. Il permet aussi d'introduire les étudiants aux nouvelles technologies comme les voitures intelligentes et autonomes. Le concours à déjà été organisé les années précédentes, le changement entre les années précédentes est l'amélioration des composants et la structure de la voiture d'année en année.

1 TABLE DES MATIERES

2 ANALYSE DU PROJET

2.1 DESCRIPTION DU SYSTEME

2.1.1 Schéma fonctionnel

Le schéma fonctionnel ci-dessous met en évidence les étapes et interactions impliqués dans le processus du système. La fonction principale est de rendre autonome une voiture, et les fonctions secondaires sont d'alimenter la voiture, suivre le parcours et éviter les obstacles mouvants (voir p.4 du rapport).

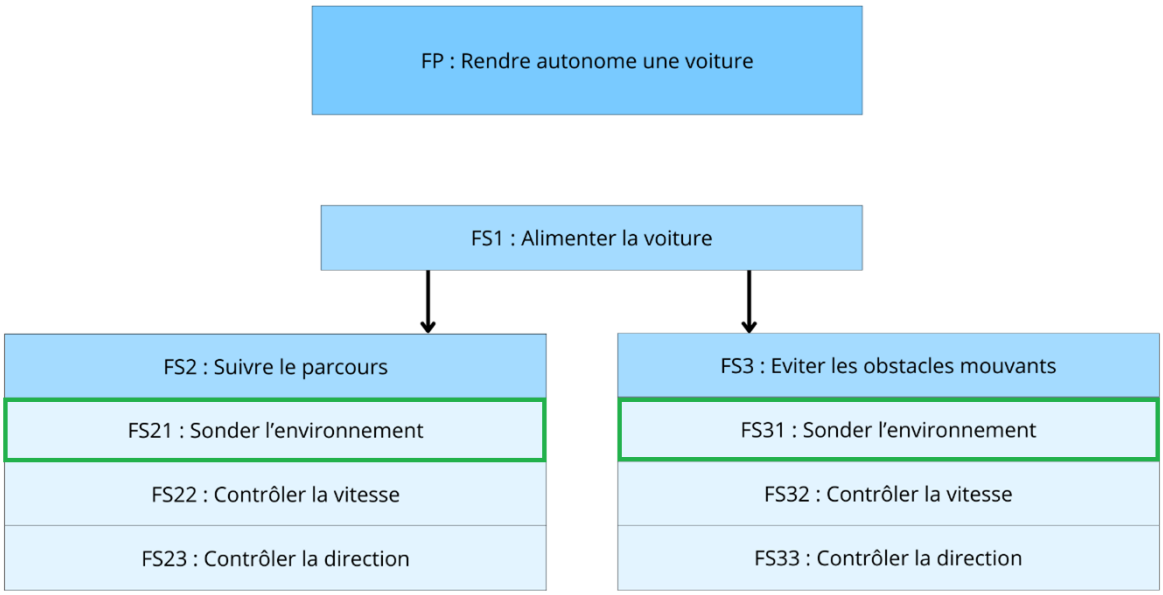


Figure 1 : Schéma fonctionnel

Nous avons enrichi la « FS21 : « Sonder l'environnement » avec une détection de couleur correspondant aux couleurs des bords du circuits de la course (rouge à gauche et vert à droite), permettant ainsi de déterminer le sens de déplacement du véhicule.

De plus, nous avons enrichi la « FS31 : Sonder l'environnement » en intégrant une détection des autres véhicules présent sur le parcours avec une évaluation de la distance, de la direction angulaire et du nombre de ceux-ci.

2.1.2 Schéma synoptique

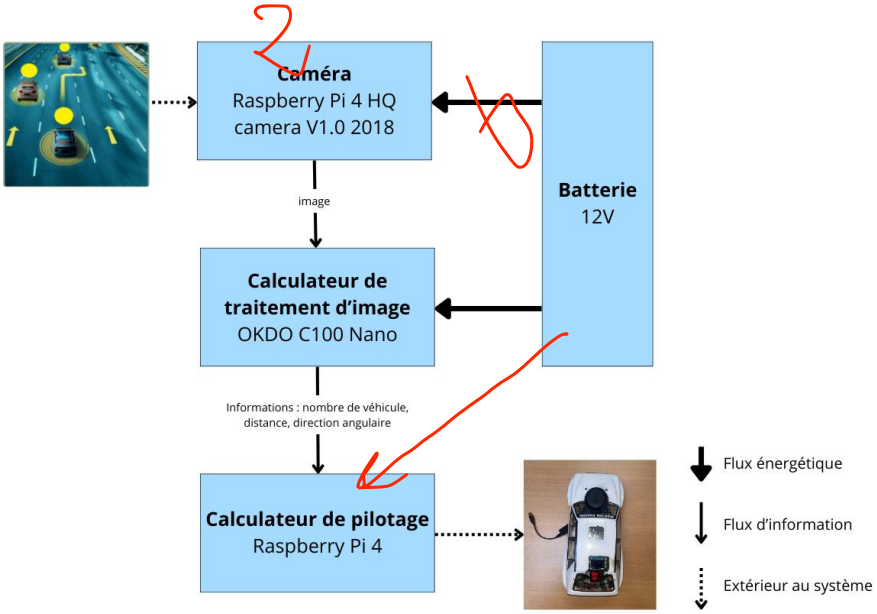


Figure 2 : Schéma synoptique

Le système est composé de 4 éléments distincts : Les caméras avant/arrière Raspberry Pi HQ camera V1.0 2018, le calculateur de traitement d'image OKDO C100 Nano, le calculateur de pilotage Raspberry Pi ainsi qu'une batterie 12V.

- **OKDO C100 Nano :** OKDO C100 Nano est une carte de développement compacte basée sur le système sur puce (SoC) ESP32-C3 de Espressif Systems. Elle offre une variété de fonctionnalités intégrées telles que le Wi-Fi, le Bluetooth LE, des broches GPIO, ainsi que des interfaces pour l'USB, l'I2C, l'UART, et plus encore.

- **Raspberry Pi 4** : Le Raspberry Pi 4 est un nano-ordinateur monocarte de la taille d'une carte de crédit, développé par la Fondation Raspberry Pi. Il est équipé d'un processeur ARM quad-core 64 bits, offrant des performances améliorées par rapport aux modèles précédents.

Les caméras et le calculateur de traitement d'image sont alimentés en 12V par la batterie. Les caméras réceptionnent les informations extérieures sous la forme d'images (Webcam Full HD Intel RealSense Depth Camera D435 1920 x 1080 Pixel). Celles-ci sont ensuite transmises au calculateur de traitement d'image (NVIDIA® OKDO C100 Nano Nano Development Kit) par liaison USB. Une fois les images traitées, et les informations (distance, direction angulaire, nombre de véhicule) extraites, ces informations sont transmises par liaison UART au calculateur de pilotage (Raspberry Pi 4).

2.2 CAHIER DES CHARGES ET SPECIFICATIONS

Dans cette partie nous allons présenter les spécifications que nous avons produit durant la partie gestion de projet. Ce tableau est créé par rapport aux différents éléments du traitement d'image et par rapport aux fonctions FS21/31que nous avons définie dans notre schéma fonctionnel.

2.2.1 Spécifications du produit

Spécifications	Quantification	Justification	Test de validation
FS21			
Détection de couleur		Réglement : Suivre le parcours	Détecter les couleurs rouges et vertes ~~en streaming~~
FS31			
Evaluation du nombre de véhicules	Tous	Règlement : nombre de participants sur le circuit = 6	Tester à l'aide de d'image de voitures, et d'impressions de voitures
Evaluation de la distance des véhicules	≃ 100 cm	Règlement : Besoin d'esquiver les concurents	Tester la portée du traitement d'image à la phase finale de test à l'aide d'un mètre
Evaluation de la direction angulaire des véhicules	≃ 10°	Règlement : Besoin d'esquiver les concurents	Tester la portée du traitement d'image à la phase finale de test à l'aide d'un programme calculant la direction angulaire

position ou ouverture ?

Figure 3 : Spécifications du produit

2.3 RESPONSABILITES ET DIAGRAMME DE GANT

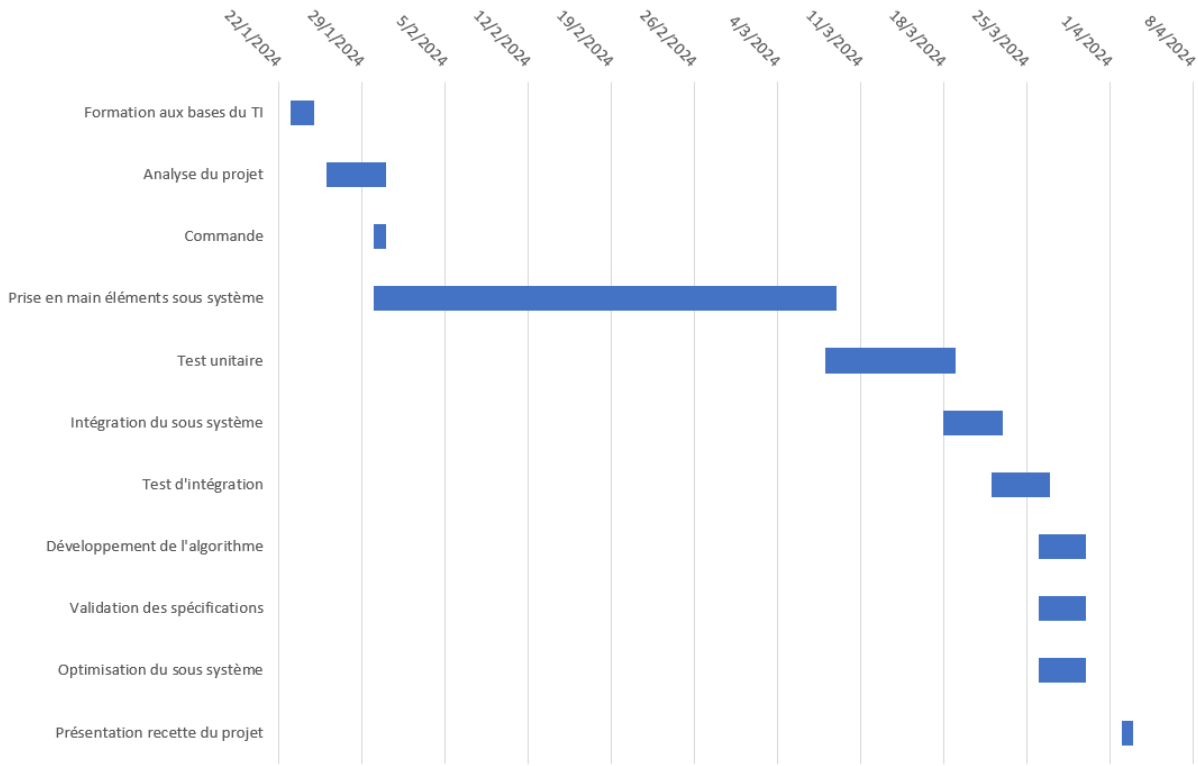


Figure 4 : Diagramme de Gantt collectif

Formation aux bases du TI : Formation aux bases du traitement d'image sous python, sur spider.
Analyse de projet : Création du cahier des charges du projet avec la recherche du protocole et des informations à transmettre, recherches des spécifications, création d'un schéma synoptique ainsi qu'un diagramme de Gantt.
Commandes : Commande des différents éléments nécessaire à la réalisation du projet que nous n'avons pas encore à notre disposition comme des caméras stéréoscopiques.
Prise en main des éléments du sous-système : Prise en main de la OKDO C100 Nano de la Raspberry, et des caméras.
Test unitaire : Test unitaire de fonction de chaque élément du sous-système (lire les éléments d'une caméra et vérifier la connexion entre la OKDO C100 Nano et la Raspberry).
Intégration du sous-système : Rendre le sous-système fonctionnel et connectés chaque élément du sous-système entre eux
Test d'intégration : Démonstration d'un traitement d'image élémentaire à partir de la caméra et transmission des coordonnées de la Raspberry.
Développement de l'algorithme : Développement de l'algorithme pour transmettre les éléments souhaités.
Validation des spécifications : Vérifier que les spécifications du cahier des charges soient bien respectées.

Optimisation du sous-système : Optimisation des derniers détails du sous-système (affichage LCD, algorithme plus rapide…)
Présentation de la recette du projet : Présentation finale du projet.

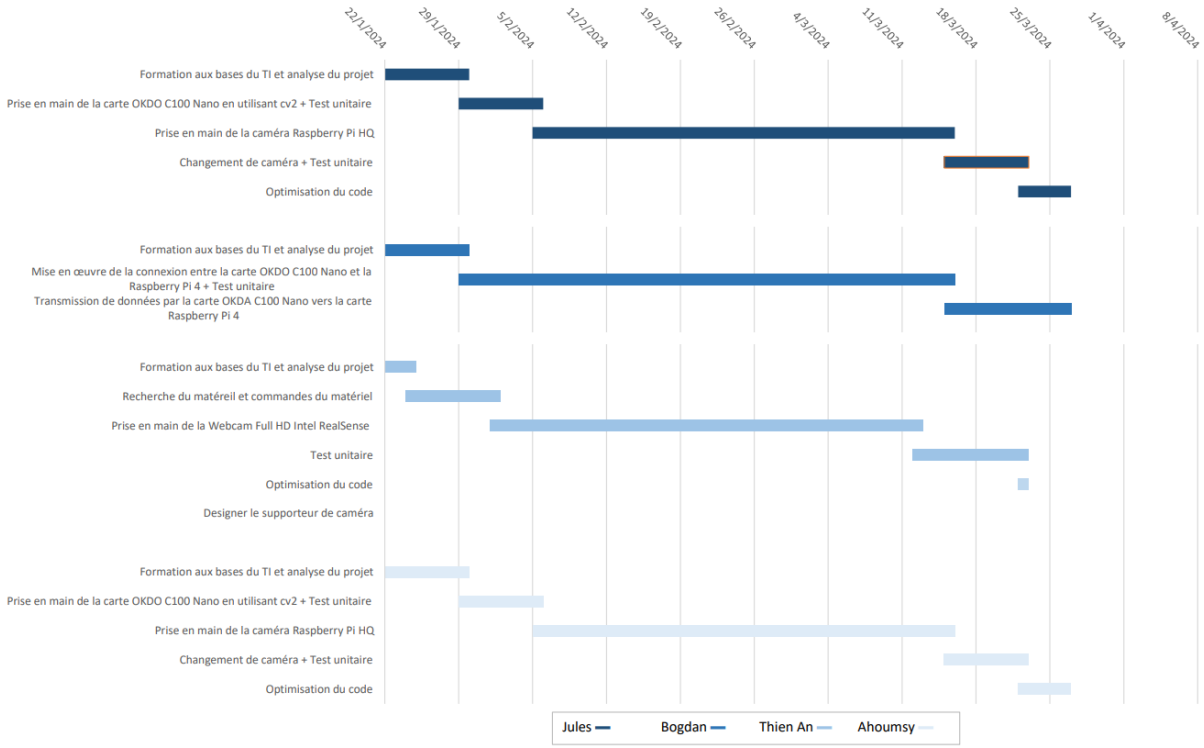


Figure 5 : Diagramme de Gantt personnel

3 FS2 – SUIVRE LE PARCOURS

La fonction FS2 « Suivre le parcours » permet à la voiture de suivre un parcours quelconque. Pour notre part, nous détaillerons uniquement la FS21 dont nous sommes responsables.

3.1 FS21 SONDER L'ENVIRONNEMENT

La fonction FS21 « Sonder l'environnement » permet de détecter le sens de course de la voiture grâce à un sous-système de détection de couleur. En effet, les bords du circuit étant rouge à gauche, et vert à droite, celui-ci compte le nombre de pixels à vert et gauche à droite et à gauche du retour caméra, et en déduit ainsi le sens de course prit par la voiture.

3.1.1 Réalisation du prototype : Détection de couleurs

3.1.1.1 Installation de l'Environnement de Travail pour OKDO C100 Nano

Figure 6 : OKDO Nano C100

Dans cette section, nous allons détailler les étapes nécessaires à la mise en place de l'environnement de travail pour la carte OKDO C100 Nano, accompagnée d'une carte SD de 32 Go.

1. **Téléchargement des Logiciels Requis :**

- Télécharger et installer le logiciel BalenaEtcher, disponible pour Linux/Mac et Windows, depuis le site officiel : BalenaEtcher.
- Télécharger le fichier nommé c100.img.xz depuis le site officiel d'OKDO : OKDO Software Hub.

2. Préparation de la Carte SD :

- Insérer la carte SD dans le lecteur de carte et la connecter au PC.
- Lancer le logiciel BalenaEtcher.
- Sélectionner le fichier c100.img.xz et la carte SD.
- Cliquer sur "Flash" et attendre que l'image soit écrite et vérifiée.
- Une fois l'opération terminée, éjecter la carte SD de l'hôte.

3. Configuration Initiale de la Carte OKDO C100 Nano :

- Insérer la carte SD dans l'emplacement prévu sur le module OKDO C100 Nano Nano.
- Connecter tous les périphériques nécessaires (clavier, souris, câble Ethernet, écran).

4. Démarrage et Configuration :

- Brancher l'alimentation.
- La LED verte à côté du connecteur micro USB s'allumera.
- Suivre les instructions à l'écran pour :
- Accepter le CLUF du logiciel NVIDIA OKDO C100 Nano.
- Sélectionner la langue, la disposition du clavier et le fuseau horaire.
- Créer un nom d'utilisateur, un mot de passe et un nom d'ordinateur.
- Accepter la taille de la partition APP (taille maximale recommandée).
- Configurer le module en mode puissance MAX.

Après avoir réalisé toutes ces étapes, voici l'écran que vous devriez avoir :

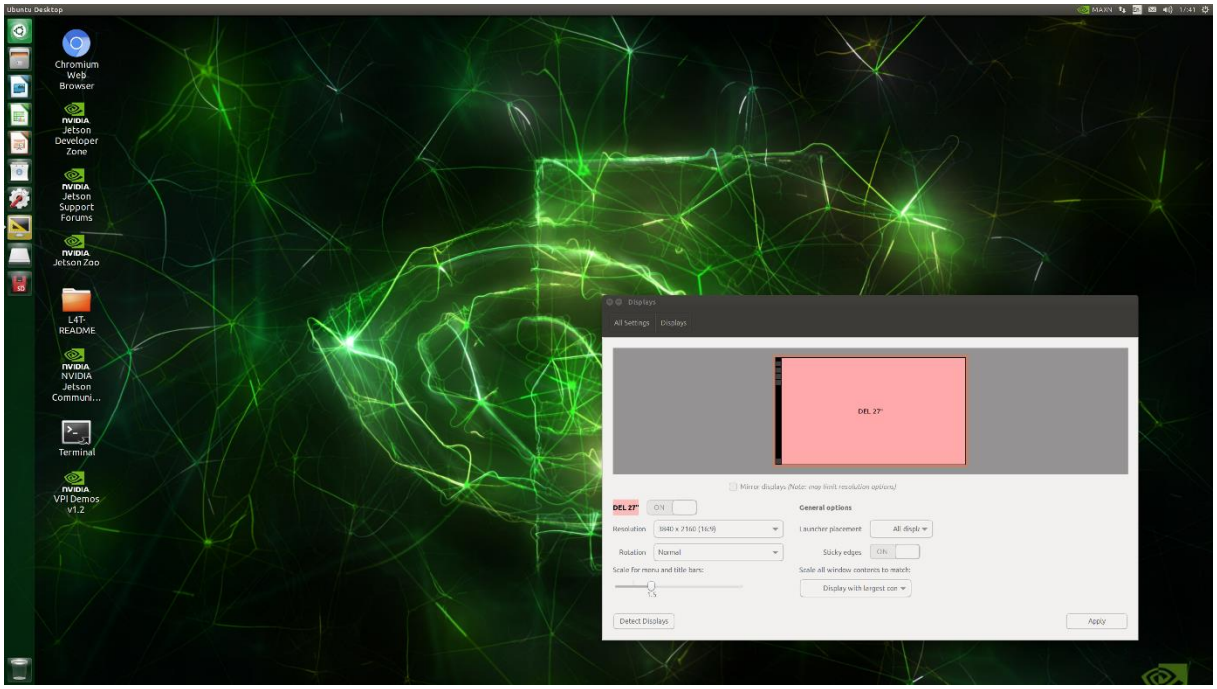


Figure 7 : L'écran après démarrage de la carte

5. **Vérification de la Version OKDO Nano C100 :**

- Ouvrir le terminal et exécuter la commande suivante pour vérifier la version de OKDO Nano C100 :

```
sudo apt-cache show nvidia-jetpack
```

- Si la version est déjà à jour (4.6.X), cette étape peut être ignorée. Sinon, exécutez ces commandes :

```
sudo apt update
sudo apt install nvidia-jetpack
```

6. **Résolution des Problèmes Potentiels :**

En cas de problèmes lors de la mise à jour, suivre les étapes ci-dessous :

- Pour le problème "Dpkg error" :

```
sudo mv /var/lib/dpkg/info/ /var/lib/dpkg/backup/
sudo mkdir /var/lib/dpkg/info/
sudo apt-get update
sudo apt-get -f install
sudo mv /var/lib/dpkg/info/* /var/lib/dpkg/backup/
sudo rm -rf /var/lib/dpkg/info
sudo mv /var/lib/dpkg/backup/ /var/lib/dpkg/info/
```

- Pour le problème "Jet_pack_probleme" :

```
sudo apt update
sudo apt-get install nvidia-container
sudo apt-get install nvidia-jetpack
sudo mv /var/lib/dpkg/backup/ /var/lib/dpkg/info/
```

Installer OpenCv (pas besoin si vous utilisez Realsense) :

1 Installer pip3 par Commencez par mettre à jour la liste des packages à l'aide de la commande suivante :

```
sudo apt update
```

2 Utiliser la commande suivante pour installer pip pour Python 3 :

```
sudo apt install python3-pip
```

3 Utiliser la commande suivante pour installer opencv

```
pip3 install opencv-python ou pip3 install opencv-contrib-python
```

3.1.1.2 Raspberry Pi 4

La Raspberry étant déjà mise en place pour la course, il n'est pas nécessaire de revenir sur son implémentation celle-ci ayant déjà été faite.

Complément de la partie sur l'implémentation de la Raspberry Pi 4 :

Pour installer les bibliothèques et activer la liaison série sur la Raspberry Pi 4, il suffit de reprendre les mêmes étapes que celle pour le faire sur la OKDO C100 Nano, expliqué dans la partie 2.1.1.1.

3.1.1.3 Logitech Webcam C920 HD Pro

Il manque les caractéristiques techniques

Figure 8 : Logitech webcam C920 HD Pro

Nous souhaitons enrichir la fonction secondaire "FS21: Sonder l'environnement" avec une détection de couleur correspondant aux couleurs des bords du circuit de la course (rouge à gauche et vert à droite), permettant ainsi de déterminer le sens de déplacement du véhicule. Pour se faire, nous utiliserons la **Logitech Webcam C920 Pro** que nous placerons sur le système afin qu'elle puisse détecter les bords du circuit, et ainsi déduire le sens de déplacement du véhicule.

Pour utiliser la webcam en streaming, il suffit d'entrer la commande dans le terminal :

```
ti@gauss08-desktop:~$ sudo su
[sudo] password for ti:
root@gauss08-desktop:/home/ti# gst-launch-1.0 v4l2src device=/dev/video0 : xvimagesink -e
```

Figure 9 : commande test de la caméra

3.1.1.4 La caméra stéréoscopie

Dans cette partie, nous abordons la connexion et l'installation de la caméra stéréoscopique Realsense D435 sur la carte OKDO C100 Nano.

Figure 10 : Caméra Realsense D435

La caméra de profondeur D435 d'Intel® RealSense™ offre un large champ de vision et une portée maximale de 10 mètres. Elle permet de mesurer la profondeur dans le domaine 3D, ce qui équivaut à la distance entre deux objets dans le domaine 2D. La connexion à la carte OKDO C100 Nano s'effectue via un câble USB-C.

Pour tester le fonctionnement de la caméra, l'application "Realsense Viewer" doit être installée sur la carte OKDO C100 Nano.

Installation des Packages Debian de Realsense:

- Enregistrement de la clé publique du serveur :

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-key
F6E65AC044F831AC80A06380C8B3A55A6F3EFCDE || sudo apt-key adv --keyserver
hkp://keyserver.ubuntu.com:80 --recv-key
```

- Ajout du serveur à la liste des référentiels :

```
sudo add-apt-repository "deb https://librealsense.intel.com/Debian/apt-repo
$(lsb_release -cs) main" -
```

- Installation du SDK :

```
sudo apt-get install librealsense2-utils
sudo apt-get install librealsense2-dev
```

Méthode Alternative d'Installation de "Realsense Viewer" :

Si la méthode précédente ne fonctionne pas, une autre méthode d'installation est disponible :

- Clonage de la bibliothèque depuis le terminal :

```
git clone https://github.com/OKDO_C100
NanoHacksNano/installLibrealsense.git
```

- Accès au répertoire "installLibrealsense" :

```
cd installLibrealsense
```

- Installation via le script :

```
./installLibrealsense.sh
```

Vérification de l'Installation :

Reconnectez la caméra Realsense et exécutez la commande suivante dans le terminal :

```
realsense-viewer
```

Si tout va bien, après avoir mis la commande « `realsense-viewer` », l'application Realsense viewer va apparaitre :

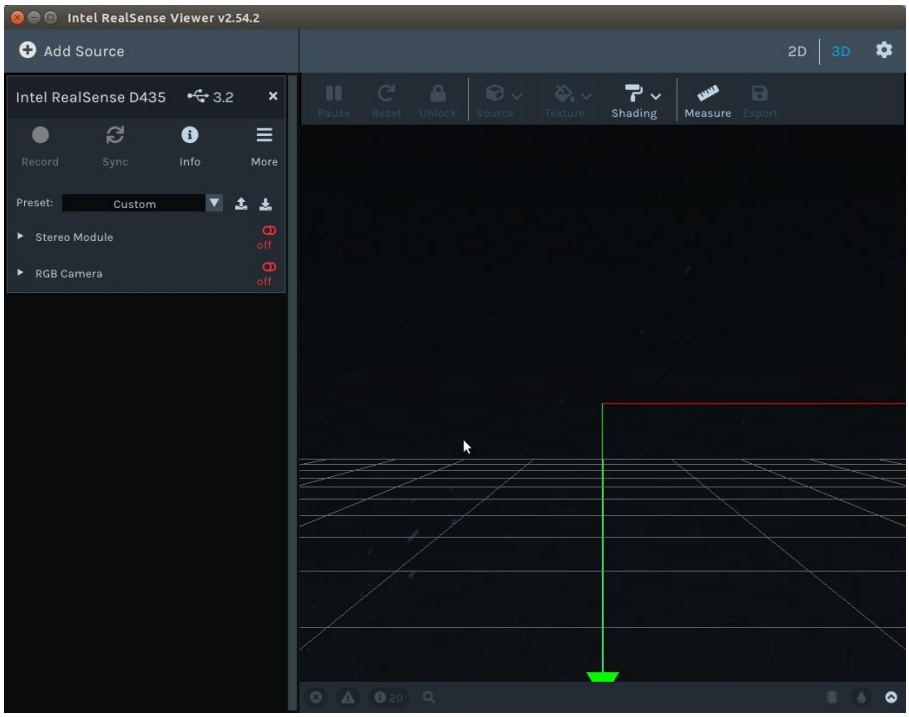


Figure 11 : L'interface de Realsense Viewer

Quand l’application démarre pour la première fois, après avoir choisi la source du périphérique, une notification de demande de mise à jour du "Firmware" apparaît. Cliquez sur "update" pour mettre à jour le "firmware" de notre caméra. Cette procédure prendra un peu de temps. Après la mise à jour du "firmware", à gauche de l’écran, il y a deux modes que nous pouvons utiliser :

Module stéréo : ce mode permet de mesurer la profondeur dans le domaine 3D ou 2D lorsqu'il est activé. Avant de passer en 3D, nous devons régler les paramètres de la caméra. Il y a deux choses que nous devons activer : "hole filling filter" et "Enable Auto Exposure". Ces deux paramètres permettent d'ajuster automatiquement le niveau d'exposition de la caméra (adapté à différentes situations) et de remplir toute la zone noire. Dans ce mode, il y a aussi la caméra infrarouge, mais nous n'en avons pas besoin dans notre projet.

Caméra RGB : qui affiche notre image en couleur. Dans les paramètres de cette partie, choisissez "Enable Auto Exposure" comme dans l'autre mode.

Lorsque nous activons les deux modes en 2D, il y a deux écrans qui affichent notre image capturée.

Dans le mode 3D, nous pouvons changer le mode d'affichage en choisissant "Texture".

Configuration de l'Environnement Python :

- Assurez-vous que "apt-get" est à jour :

```
sudo apt-get update && sudo apt-get upgrade
```

- Mettez à jour Python3 si nécessaire (version 3.6.X ou 3.7.X) :

```
sudo apt-get install python3 python3-dev
```

- Accédez au répertoire source cloné dans le terminal.
- Créez et accédez au répertoire "build" :

```
mkdir build
cd build
```

- Exécutez la commande CMake avec l'indicateur pour les bindings Python :

```
cmake ../ -DBUILD_PYTHON_BINDINGS:bool=true -DPYTHON_EXECUTABLE=[full path
to the exact python executable]
```

Utilisez la commande **which python3** pour obtenir le chemin exact vers l'exécutable Python.

- Installez les bibliothèques nécessaires :

```
make -j4
sudo make install
```

Configuration de la Variable d'Environnement PYTHONPATH : Ajoutez le chemin d'accès à la bibliothèque pyrealsense à la variable d'environnement PYTHONPATH :

```
export PYTHONPATH=$PYTHONPATH:/usr/local/lib
```

Remarque : Il est possible que la bibliothèque ne s'installe pas dans le répertoire /usr/local/lib. Consultez les journaux d'installation pour vérifier l'emplacement exact de l'installation. Dans certains cas, la bibliothèque peut être installée dans /usr/local/OFF.

3.1.1.5 Support de caméra

Dans cette partie nous allons créer le support qui nous permettons de fixer les caméras sur la voiture autonome. Ce support a 2 parties :

+ Pied de support

+ La plage de connexion de caméras

Pour créer le support de caméra, nous allons utiliser SolidWorks qui nous permet de créer un 3D modèle.

Attention : Avant de designer votre modèle, vérifier les matériels dans magasine (la taille de vis, ...)

Le pied de support :

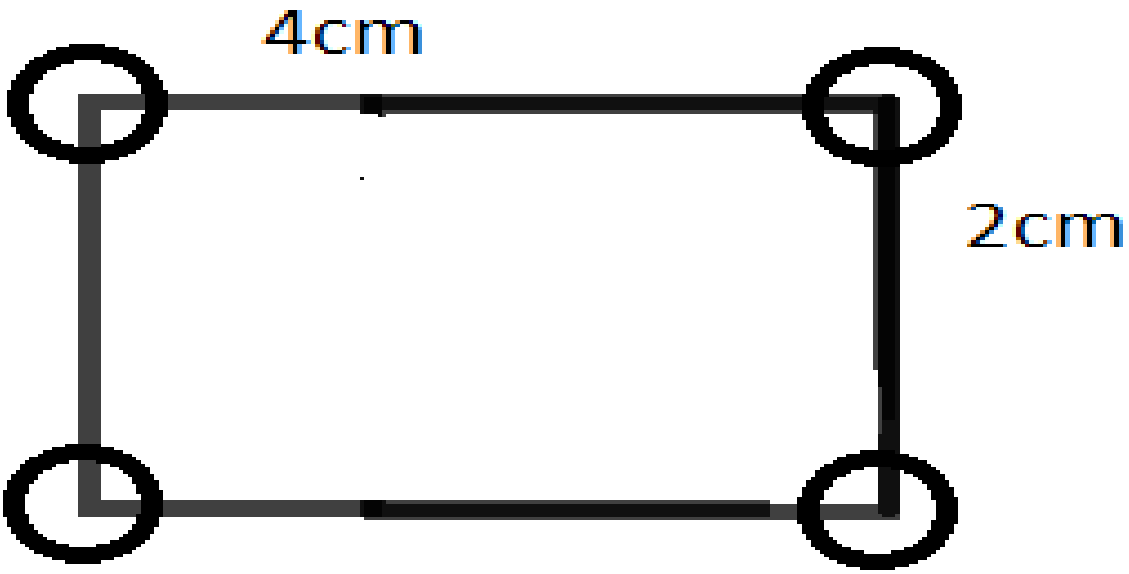


Figure 12 : Schéma du pied de support

Les positions de fixation du pied de support forment le rectangle d'une longueur de 4 cm et d'une largeur de 2 cm. Chaque trou de vis a un rayon de 0,25 cm.

Pour créer le pied, nous passons les séquences des étapes :

1. Dans SolidWorks, choisissiez nouvelle (Crtl+N) et clique sur pièce.
2. Choisissiez une face pour désigner et cliquez esquisse pour commencer de designer l'esquisse de notre modèle : un rectangle avec la longueur de 54 mm et le largeur de 32 mm. Utilisez la "Cotation Intelligente" pour modifier la longueur d'une partie d'esquisse
3. Quittez l'esquisse et choisissez fonction : Bossage/ Base extrudé. Modifiez la hauteur de modèle à 10 mm.
4. Cliquez droite sur la plus grande face de modèle et choisissiez "normal à".
5. Cliquez esquisse pour créer le pilier de connexion.
6. Créez un cercle au millier de la face avec le rayon de 5mm.

7. Quittez l'esquisse et choisissez fonction : Bossage/ Base extrudé. Modifiez la hauteur du pilier (6 – 7mm)
8. Choisissiez la face opposée et cliquez sur assistance pour le perça.
9. Choisissiez le type de trou de la vis et sa taille (vis M3 dans ANSI métrique)
10. Cliquez à "position" pour placer le perça. Modifiez sa position par "Cotation Intelligente".
11. Choisissiez la face avec le pilier et faites perça comme étape précédent avec la taille M2.5
12. Option : pour le pied plus beau, choisissiez "congés" et choisissez les faces que nous pouvons faire congés.

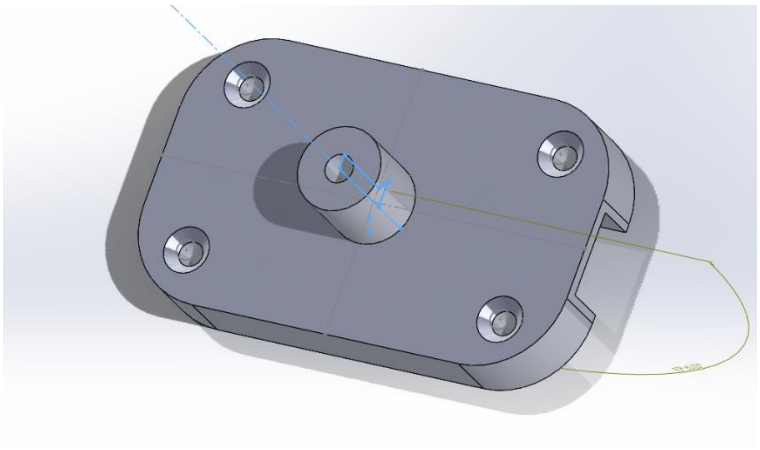


Figure 13 : Pied de support

La plage de connexion de caméras, nous passons les séquences des étapes très similaire, mais la largeur de plage est 70mm, la longueur de plage est 91mm, les deux piliers de connexion de caméra (rayons de 10mm) et les trous pour le câble USB (17mm de longueur, 8mm de largeur et créer par enlèvement de matière) et un trou pour la vis (3.4mm de diamètre).

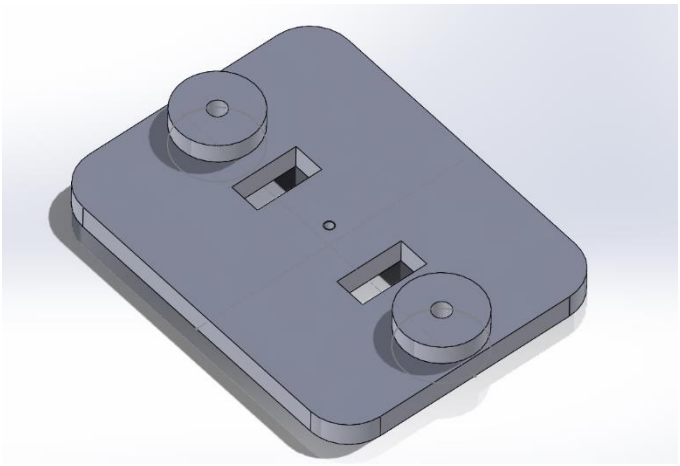


Figure 14 : Plage de connexion

Figure 15 : Intégration du pied de support avec le système

3.1.2 Test du prototype : Détection de couleurs

Dans cette partie du Rapport, nous réaliserons les tests unitaires relatifs à chaque élément du sous-système.

3.1.2.1 Test unitaire de la Raspberry Pi 4 (Complément)

Pour le test unitaire de la Raspberry Pi 4, l'objectif sera d'observer un signal carré à l'oscilloscope via liaison UART, pour s'assurer du bon fonctionnement de l'UART sur la carte.

On commencera par importer les bibliothèques nécessaires : "serial" pour utiliser le port série et "time" pour pouvoir crée des attentes manuelles.

```
import serial
import time
```

Par la suite, on configurera le port série en utilisant le port 'ttyS0', et on utilisera un baudrate de 9600 qui est celui de la machine.

```
# Configuration du port série
serial_port = '/dev/ttyS0'
baud_rate = 9600

# Ouvrir le port série
ser = serial.Serial(serial_port, baud_rate)
```

Enfin, dans une boucle infinie, on enverra toutes les demi-secondes en alternance un niveau haut ou un niveau bas à l'aide de la fonction '.write()' de la bibliothèque "serial".

```
while True:
    # Envoyer un niveau haut (1)
    ser.write(b'1')
    time.sleep(0.5)  # Attendre 0.5 seconde
    # Envoyer un niveau bas (0)
    ser.write(b'0')
    time.sleep(0.5)  # Attendre 0.5 seconde
```

On connectera les broches Ground(6) et GPIO14-TXD(8) à l'entrée de l'oscilloscope pour observer le signal suivant :

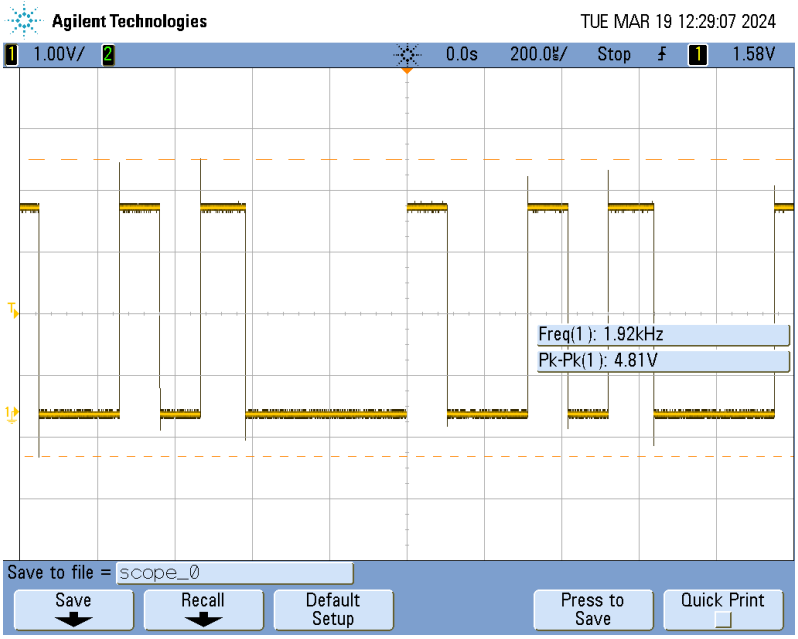


Figure 16 : Signal réceptionné

3.1.2.2 Test unitaire de la OKDO C100 Nano

Afin de tester la carte OKDO C100 Nano, nous avons réalisé un programme de détection de contours à l'aide de la librairie cv2.

- Télécharger une photo dans le dossier voulu

Dans le terminal, se placer dans le même dossier que les photos et taper la ligne de commande :

gedit test_unitaire.py

Puis entrez le code ci-dessous :

```
#!/usr/bin/python
# -*- coding utf-8 -*-

import cv2

img = cv2.imread('/home/ti/Images/lena.png', cv2.IMREAD_GRAYSCALE)

import numpy as np

arr = np.array(img)
image = np.zeros((512,512), dtype = np.uint8)

image = cv2.Canny(img, 50, 140)

cv2.imshow('image', img)
cv2.imshow('image ligne detecte', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Figure 17 : Détection de contours

Revenez sur le terminal et fermer la fenêtre gedit : ctrl + c

Lancez le code avec la commande : python3 test_unitaire.py

Ce code devrait afficher les deux fenêtres ci-dessous :

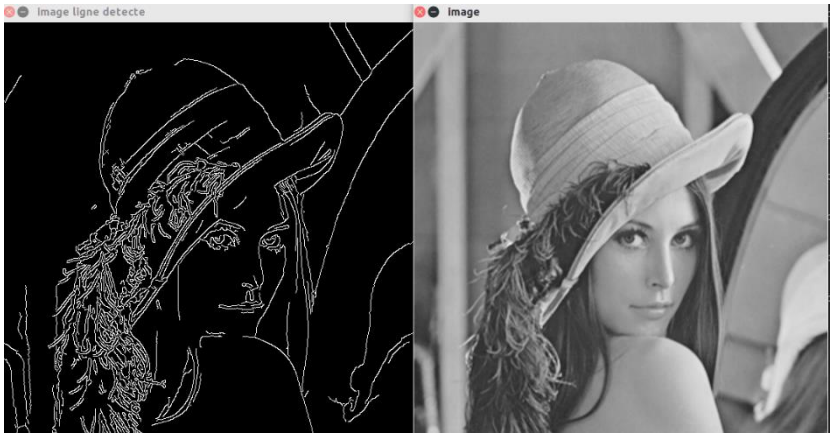


Figure 18 : Résultat de la détection de contours

Pour la communication de l'UART avec la OKDO C100 Nano, le test unitaire sera le même que celui de la Raspberry. La seule différence et que l'on utilisera le port 'ttyTHS1'. On pourra donc se référer à la partie 2.1.1.1. .

3.1.2.3 Test unitaire de la Logitech Webcam C920 HD Pro

Afin de détecter le sens de course du véhicule, nous avons réalisé un code détection de couleur en python que nous avons programmé sur la carte OKDO C100 Nano.

Commençons par configurer la capture d'image par la webcam :

Nous importons la bibliothèque cv2 OpenCV, qui est au préalable installé sur la carte OKDA C100 Nano.

```
1   #Importation de la bibliothèque OpenCV
2   import cv2
```

Nous sélectionnons ensuite la webcam Logitech pour la lecture vidéo, que nous affecterons à la variable cap. Ici, la valeur 0 indique la première webcam disponible.

```
4   # Initialiser la capture vidéo à partir de la webcam :
5   cap = cv2.VideoCapture(0)
```

Pour des soucis d'efficacité et pour répondre aux spécifications, nous changeons la résolution de la lecture vidéo étant de 3280x2464 pour une résolution HD de 1920 pixels pour 1080. La résolution HD nous permet de détecter les couleurs avec plus de précision. Cependant, en HD, nous passons de 30 frames (images par secondes) à 5 frames.

```
12  # Initialiser la résolution :
13  cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1920)
14  cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 1080)
```

La lecture vidéo étant optimale, nous pouvons passer à l'exploitation de celle-ci pour la détection de couleurs. Nous commençons par extraire chaque frame à l'aide la fonction cap.read(), que nous mettrons dans une boucle while(True) afin que celle-ci s'exécute en continue.

```
16  # Lire continuellement les images de la webcam :
17  while True:
18      ret, frame = cap.read()
```

Nous divisions ensuite le frame en deux parties (droite et gauche) afin de pouvoir étudier chacune d'entre elles de manière distincte et indépendante. Ici, la fonction frame.shape renvoie la hauteur et la largeur de chaque frame, soit 1920 et 1080, puis créer deux régions de la frame, left_half et right_half.

```
23      # Diviser l'image en deux moitiés :
24      height, width, _ = frame.shape
25      left_half = frame[:, :width // 2]
26      right_half = frame[:, width // 2:]
```

Afin d'obtenir une détection des couleurs efficaces, nous choisissons de convertir les frames en format HSV (Hue, Saturation, Value) au lieu du format RGB (Red, Green, Blue). En effet, le format HSV est celui communément utilisé dans la détection de couleur

```
28      # Convertir les images en format HSV pour une meilleure détection des couleurs :
29      left_hsv = cv2.cvtColor(left_half, cv2.COLOR_BGR2HSV)
30      right_hsv = cv2.cvtColor(right_half, cv2.COLOR_BGR2HSV)
```

Nous définissons ensuite les plages de vert et de rouge que nous voulons étudier à l'aide d'un HSV Color Picker (https ://www.selecolor.com/en/hsv-color-picker/). Nous avons choisi ces valeurs expérimentalement en fonction des différentes observations du circuit :

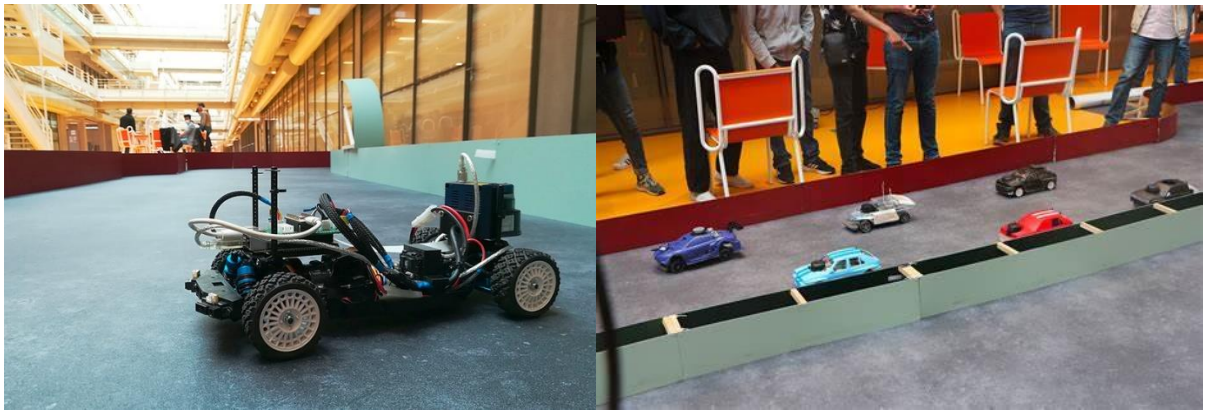


Figure 16 : Photos du circuit

Le code HSV du bord rouge correspond à (358, 77, 44) et celui du bord vert correspond à (171, 26, 74). Nous sélectionnons des plages de couleur car la luminosité peut faire varier le code HSV d'une teinte. Pour déterminer le vert et rouge foncé ou clair, nous avons utilisé l'outil Color Picker PowerToys, qui détermine la teinte claire et foncé d'une couleur.

```
32    # Définir les plages de couleur pour le vert et le rouge en HSV :
33       lower_green = (170, 26, 44)
34       upper_green = (188, 255, 244)
35       lower_red1 = (355, 75, 20)
36       upper_red1 = (188, 43, 48)
```

Après avoir déterminer les plages de couleurs que nous voulons étudier, nous filtrons les pixels appartenant à ces plages grâce à la fonction cv2.inRange(). En sortie de ces filtres, les pixels contenus dans ces plages auront la valeur binaire 255 (blanc). Les pixels n'étant pas situés dans ces plages se verront attribué la valeur binaire 0 (noir).

```
38      # Filtrer les pixels verts et rouges de chaque côté :
39      mask_green_left = cv2.inRange(left_hsv, lower_green, upper_green)
40      mask_green_right = cv2.inRange(right_hsv, lower_green, upper_green)
41      mask_red1_left = cv2.inRange(left_hsv, lower_red1, upper_red1)
42      mask_red1_right = cv2.inRange(right_hsv, lower_red1, upper_red1)
```

Nous comptons ensuite les pixels blancs résultants des filtres de chaque côté du frame à l'aide de la fonction cv2.countNonZero(), qui compte les pixels non nuls (blancs) en sortie des filtres.

```
    # Compter les pixels verts et rouges de chaque côté :
    green_left_count = cv2.countNonZero(mask_green_left)
    green_right_count = cv2.countNonZero(mask_green_right)
    red_left_count = cv2.countNonZero(mask_red1_left)
    red_right_count = cv2.countNonZero(mask_red1_right)
```

A l'aide d'une boucle condition if, on établit le sens de déplacement avec une variable éponyme de type bool. Si le nombre de pixel vert à droite de la frame est supérieur à celui à gauche, et que le nombre de pixel rouge à gauche de la frame est supérieur à celui à droite, alors le système se déplace dans le bon sens et la variable sens prend la valeur True. Dans le cas inverse, le véhicule se déplace dans le mauvais sens, et la variable sens prend la valeur False.

```
    # Comparer les côtés pour déterminer le sens de la course :
    if green_right_count >= green_left_count and red_left_count >= red_right_count:
        sens = True
        print("Sens de la course : ", sens)
    else:
        sens = False
        print("Sens de la course : ", sens)
```

Finalement, on affiche les résultats ainsi que le sens de course sur le terminal.

```
    # Afficher les résultats :
    print("Nombre de pixels verts - Gauche :", green_left_count, "Droite :", green_right_count)
    print("Nombre de pixels rouges - Gauche :", red_left_count, "Droite :", red_right_count)
```

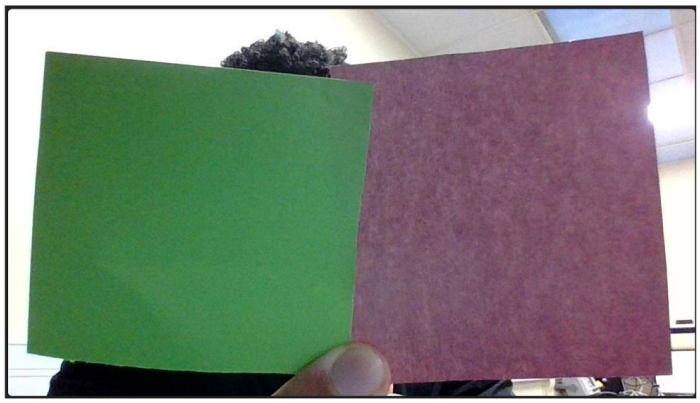


Figure 17 : Retour vidéo de la Webcam Logitech

```
sens de la course =  True
Nombre de pixels verts - Gauche : 909 Droite : 909
Nombre de pixels rouges - Gauche : 82213 Droite : 82213
```

Figure 18 : Retour du Terminal

3.1.3 Test d'intégration du prototype : Détection de couleurs

Pour le test d'intégration de la détection de couleurs, il faut récupérer les variables "sens", "green_left_count", "green_right_count", "red_left_count" et "red_right_count" issue du code expliqué dans la partie 2.1.2.3, et les envoyés vers la Raspberry Pi 4 via liaison UART. Il faudra d'abord écrire un code pour l'émetteur, c'est-à-dire la carte OKDO C100 Nano. Pour cela, on ajoutera les lignes de code suivante au code déjà expliqué dans la partie 2.1.2.3 :

On commence par importer les bibliothèques nécessaires pour la transmission de données : "serial" qui servira à envoyer des données via liaison série et "time" qui permettra de mettre une attente dans l'envoie de données.

```
import serial
import time
```

On initialise le port série par la suite. Sur la Okdo OKDO C100 Nano Nano, on utilisera le port 'ttyTHS1', car le port 'ttyS0' n'est pas utilisable. On définit le baudrate à 9600 car c'est celui donnée par les spécifications de la machine.

```
ser = serial.Serial('/dev/ttyTHS1', baudrate=9600, timeout=1)
```

Dans la boucle infinie, on récupère les valeurs des 5 variables voulues que l'on va stocker dans une variable 'data'. Puis on l'enverra à l'aide de la fonction de '.write()' de la bibliothèque "serial".

```
data = f"{sens}, {green_left_count}, {green_right_count}, {red_left_count}, {red_right_count}\n"
ser.write(data.encode())
```

Enfin, encore dans la boucle infinie, on ira mettre une attente de 1 seconde pour que l'envoie puisse faire toutes les secondes. Le seul bémol sera que sur la frame, l'affichage de ce qu'observe la webcam sera beaucoup plus lent.

```
time.sleep(1)
```

Par la suite, on écrira un code pour recevoir les données envoyées par la OKDO C100 Nano sur la Raspberry et pour lire ces données en question. Pour ce faire, sur la Raspberry on écrira un code en python qui où l'on y importera la bibliothèque "serial".

On initialise le port série. On utilise cette fois-ci le port 'ttsyS0' qui est disponible sur la Raspberry Pi 4. On utilisera le même baudrate que celui de la OKDO C100 Nano. En réalité, les deux machines utilisent le même baudrate de 9600.

```
ser  =  serial.Serial('/dev/ttyS0', baudrate = 9600, timeout = 1)
```

On crée une variable 'data' qui stockera les valeurs reçues venant de la OKDO C100 Nano grâce à la fonction '.readline()'.

```
data = ser.readline().decode().strip()
```

Si on reçoit bien des données, c'est-à-dire que la variable data se met à jour, alors on utilise la fonction 'map()' pour convertir les données reçues en une structure appropriée. Après avoir lu les données série à partir du port série, les données sont sous forme de chaîne de caractères. La fonction 'split()' est utilisée pour diviser cette chaîne en parties distinctes en utilisant un délimiteur, dans ce cas, la virgule.

La fonction 'map()' permet ensuite de transformer les données reçues (qui sont initialement des chaînes de caractères) en valeurs numériques appropriées pour un traitement ultérieur.

Au final, on affiche sur le terminal les valeurs de chaque variable qui ont été transmises en temps réel.

```
if data :
    sens, green_lef_count, green_right_count, red_left_count, red_right_count = map(int, data.split(','))
    print("Sens de la course :" , sens)
    print("Pixels verts gauche : ", green_right_count, "droite : ", green_right_count)
    print("Pixels rouge gauche : ", red_right_count, "droite : ", red_right_count)
```

Enfin, on ira relier les pins Ground(6) de la OKDO C100 Nano à la Raspberry, et la pin GPIO14-TXD(8) de la OKDO C100 Nano à la pin GPIO15-RXD(10) de la Raspberry pour assurer la connxion via l'UART des deux appareils. On pourra donc aussi se référer à leurs datasheets, elles sont le même Pin planning.

4 FS3 EVITER LES OBSTACLES MOUVANTS

La fonction FS3 « Eviter les obstacles mouvants » permet à la voiture d'esquiver les véhicules environnants. Pour notre part, nous détaillerons uniquement la FS31 dont nous sommes responsables.

4.1 FS31 SONDER L'ENVIRONNEMENT

La fonction FS31 « Sonder l'environnement » permet de détecter les véhicules environnants. Nous procèderons à l'aide d'un programme par apprentissage utilisant un réseaux neuronal dédié à la détection et à la reconnaissance d'objets.

4.1.1 Réalisation du prototype

Dans cette partie, nous avons 2 stratégiques pour éviter l'obstacle pour le carmera Realsense D435/D435i :

+ Détecter les obstacles par le contour

+ Détecter les obstacles en utilisant YOLOv3 ou YOLOv3-Tiny

4.1.1.1 Détecter les obstacles par le contour

Dans cette stratégie, nous utilisons la fonction : cv2.findContour de “opencv” pour trouver le contour des obstacles comme les voitures ou les blocs.

Pour cette stratégique, il y a 2 méthodes : Détecter le contour en utilisant la caméra couleur ou Détecter le contour en utilisant la caméra profondeur

Caméra couleur :

1. Utiliser application de Realsense Viewer et prendre une photo couleur de voiture ou l’obstacle en mode 2D.
2. Utiliser application de supprimer arrière-plan et l’autre chose que nous n'avons pas besoin sur l’internet.
3. Faire une détection de contour sur cette photo :

```
image_sample = cv2.imread('/home/ti-2024/Desktop/projet-ti/image/VSB2')


# Convert image to HSV color space

hsv = cv2.cvtColor(image_sample, cv2.COLOR_BGR2HSV)


# Define lower and upper bounds for the background color (white)

lower_white = np.array([90, 10, 40], dtype=np.uint8)

upper_white = np.array([225, 255, 200], dtype=np.uint8)


# Create a mask for the background

mask_background = cv2.inRange(hsv, lower_white, upper_white)


# Create a black canvas of the same size as the input image

mask = np.zeros_like(image_sample[:, :, 0])


# Find contours in the background mask

contours_sample, _ = cv2.findContours(mask_background, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)


sample_contour=max(contours_sample, key=cv2.contourArea).
```

4. Traiter l’image continue que nous obtenons directement par la caméra en utilisant cv2.cvtColor() et cv2.Canny() pour changer la photo en gris.
5. Utiliser la fonction cv2.findContours() pour détecter le contour l’image.
6. Comparer les valeurs de contour d’image direct avec les valeurs de contour de “image_sample” :

match=cv2.matchShapes(contour1,contour2, cv2.CONTOURS_MATCH_I1, 0.0)

7. Calculer la similarité entre 2 images : similarity = 1 –match
8. Si la similarité est supérieure que 70%, utiliser la fonction cv2.drawContours pour mettre le contour autour de l'objet.

Malheureusement, cette méthode a un gros inconvénient : il ne peut pas distinguer les choses dans l'image avec la voiture ou obstacle que nous voulons.

Caméra profondeur :

En utilisant la caméra profondeur, nous pouvons séparer les voitures ou les obstacles avec l'autre objet :

1 Utiliser application de Realsense Viewer et prendre une photo profondeur de voiture ou l'obstacle en mode 2D.
2 Utiliser application de supprimer les parties que nous n'avons pas besoin sur l'internet.
3 Faire une détection de contour sur cette photo comme la troisième étape de partie caméra couleur :
4 Traiter l'image continue que nous obtenons directement en utilisant la mode profondeur de la caméra realsense (changer l'image en gris) :

depth_image=np.asanyarray(depth_frame.get_data())

depth_colormap= cv2.applyColorMap(cv2.convertScaleAbs(depth_image, alpha=0.03),cv2.COLORMAP_HSV)

gray_frame =cv2.cvtColor(depth_colormap, cv2.COLOR_BGR2GRAY)

_, foreground_mask =cv2.threshold(gray_frame, 100 , 255, cv2.THRESH_BINARY)

background_removed = cv2.bitwise_not(foreground_mask)

edges = cv2.Canny(background_removed, 100 ,120)

5 Utiliser la fonction cv2.findContours() pour détecter le contour l'image.
6 omparer les valeurs de contour d'image direct avec les valeurs de contour de "image_sample" :

match=cv2.matchShapes(contour1,contour2, cv2.CONTOURS_MATCH_I1, 0.0)

7 Calculer la similarité entre 2 images : similarity = 1 –match
8 Si la similarité est supérieure que 70%, utiliser la fonction cv2.drawContours pour mettre le contour autour de l'objet.

4.1.1.2 Détecter les obstacles en utilisant YOLOv3 ou YOLOv3-Tiny

YOLOv3 (You Only Look Once version 3) est un modèle de réseau de neurones convolutifs utilisé pour la détection d'objets en temps réel dans des images ou des vidéos. Il est connu pour sa rapidité et son efficacité, ce qui en fait un choix populaire pour de nombreuses applications d'intelligence artificielle et de vision par ordinateur.

Le principe de base de YOLOv3 est de diviser une image en une grille de cellules et de prédire les boîtes englobantes (bounding boxes) ainsi que les classes des objets présents dans chaque cellule. Contrairement à d'autres approches qui utilisent des régions d'intérêt (ROIs) et des étapes de traitement multiples, YOLOv3 utilise une seule passe de convolution pour prédire les boîtes englobantes et les classes simultanément.

Nous avons décidé d'utiliser YOLOv3-Tiny car cette version est plus légère et lorsque l'on utilisait YOLOv3, le code nous envoyait les informations toutes les 5 secondes environ, ce que l'on a jugé trop lent.

Pour faire fonctionner le code, on a besoin des fichiers suivants :

coco.names

yolov3-tiny.cfg

yolov3-tiny.weights

Télécharger les fichiers sur : https://github.com/pjreddie/darknet/blob/master/cfg/yolov3-tiny.cfg

Importer le code de détection sur GitHub : https://github.com/dovanhuong/dev_realsense_yolo_v3_2d/blob/main/script_dev_realsense_yolo_v3_2d.py

Remplacer ces lignes de code afin de pouvoir utiliser l'algorithme de YOLOv3-Tiny dans votre code :

classesFile = "coco.names"

With open(classesFile,"rt" as f :

    Classes=f.read().rstrip('\n').split('\n')

modelConfiguration = "yolov3-tiny.cfg"

modelweight = "yolov3-tiny.weights"

net =cv2.dnn.readNetFromDarkNet(modelConfiguration, modelweight)

net.setPreferableBackend(cv2.dnn.DNN_BACKEND_OPENCV)

net. SetPreferableTarget(cv2.dnn.DNN_TARGET_OPENCV)

Dans notre application, nous voulons détecter seulement les voitures. Donc dans la fonction drawPredicted modifier le code :

```
if classes:
        assert(classId < len(classes))
        label = '%s' %(classes[classId])
        if label == 'car':
            print(f"Objet détecté: {label}, Distance: {distance}, Position:
{x}, {y}")
```

4.1.1.3 Raspberry Pi 4

Se référer à la partie 2.1.1.2.

4.1.2 Test du prototype : Détection des véhicules environnants

Dans cette partie, nous présenterons les différents tests du sous-système.

4.1.2.1 Test unitaire de la Raspberry Pi 4

Se référer à la partie 2.1.2.1.

4.1.2.2 Test unitaire du protocole de détection des obstacles en utilisant YOLOv3

Afin de tester si le protocole de détection de voiture fonctionnait bien, nous avons créer un banc de test correspondant à l'environnement de la course voiture autonome de l'ENS Paris-Saclay.

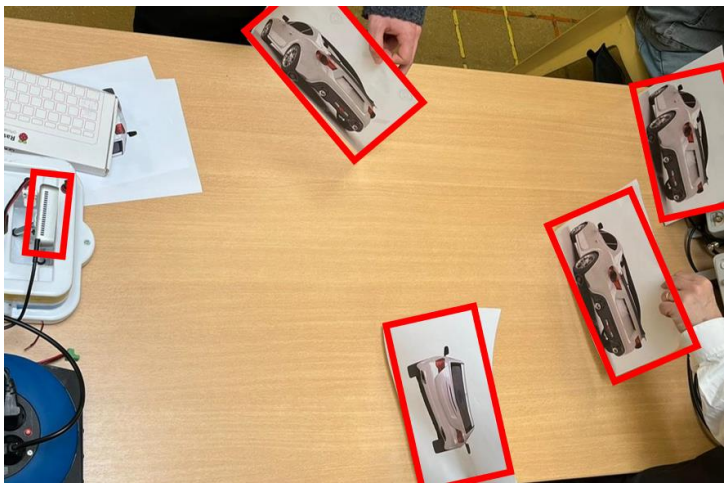


Figure 19 : Banc de test du prototype

Ici, la table faisait 90cm de largeur, soit la même largeur que la piste lors de la course voiture autonome. De plus nous pouvions, à l'aide d'images de voitures, tester combien de voitures pouvions-nous détecter en même temps. L'angle maximum de détection et la distance maximale.

- Angle maximum : +/- 18°
- Distance maximale : 98cm
- Nombre de voiture maximum détecté : 4

Le nombre de voitures maximum détectées simultanément dépendait du nombre de voiture que l'on pouvait mettre dans le champ de vision de la caméra. Ici, nous n'avons pas réussi à en mettre plus de 4.

En comparaison aux spécifications, l'angle maximum est largement suffisant. Pour la distance maximale, nous avions mis 1m minimum. Mais après avoir travaillé sur le projet nous nous sommes

rendu compte que détecter des voitures à plus d'1m de distance n'avait aucuns intérêts pour la course. Et finalement pour le nombre de voiture, nous respectons les spécifications car nous détectons toutes les voitures, dans le champ de vision de la caméra.

4.1.3 Test d'intégration du prototype : détection des véhicules environnants

Se référer à la partie 4.1.2.2 .

non parce qu'on va jusqu'à la raspberry

.

5 TESTS DE VALIDATION DU PROTOTYPE : DETECTION DE COULEURS

tous le prototype = intégrer la partie 6 de ce rapport

Dans cette partie, nous vérifierons que le prototype répond aux spécifications, et nous verrons les modifications que nous pourrons apporter au sous-système afin d'améliorer celui-ci.

5.1 RESULTATS DES TESTS DE VALIDATION DU PROTOTYPE : DETECTION DE COULEURS

Finalement, le prototype détecte bien les pixels, et en déduit bien le sens de course.

```
sens de la course =  True
Nombre de pixels verts - Gauche : 997 Droite : 997
Nombre de pixels rouges - Gauche : 83974 Droite : 83974
```

Figure 20 : Validation détection de couleurs

5.2 CONCLUSIONS TECHNIQUES SUR LE PROTOTYPE : DETECTION DE COULEURS

ici il faut revenir sur les spécifications : mais peut-être sont-elles incomplètes ?

Le prototype répond aux spécifications. Cependant, on pourrait l'améliorer en réalisant un programme de calibration de couleurs, afin de l'affiner.

.

6 TESTS DE VALIDATION DU PROTOTYPE : DETECTION DES VEHICULES ENVIRONNANTS

Dans cette partie, nous vérifierons que le prototype répond aux spécifications, et nous verrons les modifications que nous pourrons apporter au sous-système afin d'améliorer celui-ci.

6.1 RESULTATS DES TESTS DE VALIDATION DU PROTOTYPE : DETECTION DES VEHICULES ENVIRONNANTS

Finalement, le sous-système détecte 5 véhicules théoriquement mais 3 véhicules physiquement à cause de la largeur du circuit. Il a une portée de 98 cm, une amplitude de 18° avec une incertitude de 5 mm.

```
Objet détecté: car, Distance: 1.2260000705718994, Position: 157, 303
Objet détecté: car, Distance: 0.9800000190734863, Position: 323, 287
```

Figure 21 : Validation détection d'obstacles environnants

6.2 CONCLUSIONS TECHNIQUES SUR LE PROTOTYPE : DETECTION DES VEHICULES ENVIRONNANTS

Le sous-système répond aux spécifications. Nous pourrions l'améliorer en réduisant le temps de réponse, et ceci, en optimisant le programme.

7 RETOUR D'EXPERIENCE SUR LA GESTION DE PROJET

7.1 ANALYSE DE L'ECART ENTRE LE GANTT ETABLI EN DEBUT DE PROJET ET LE GANTT REEL EN FIN DE PROJET

Nous allons observer les différences entre le diagramme créé en début de projet qui reflète la perception du projet avant de le commencer et le diagramme créé à la fin qui montre la gestion des tâches durant le projet.

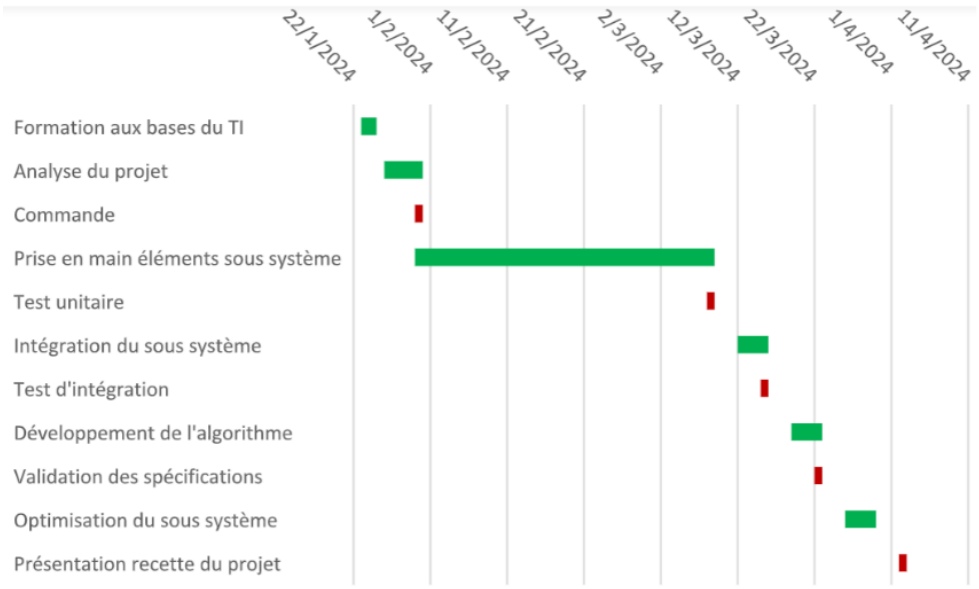


Figure 22 : Diagramme de Gantt initial

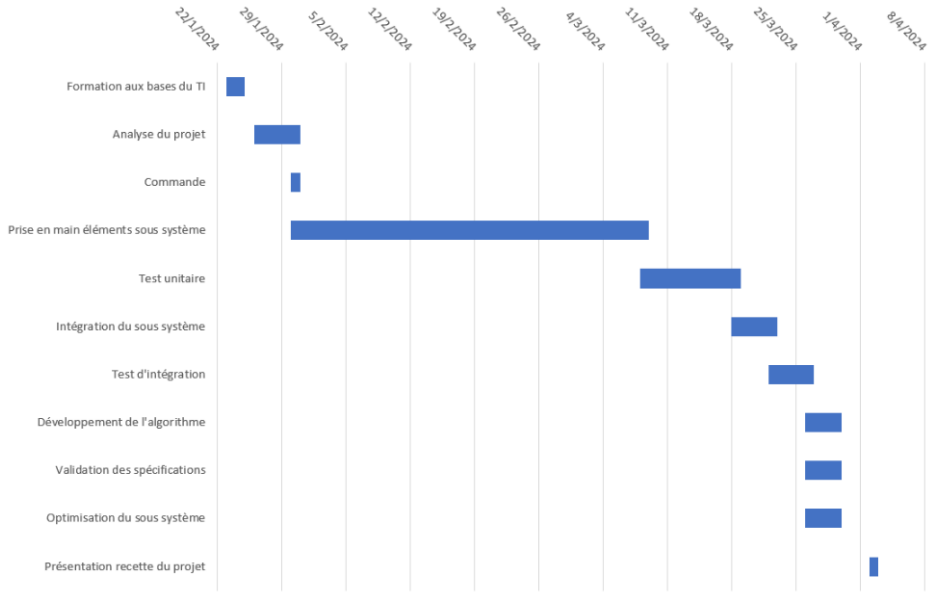


Figure 23 : Diagramme de Gantt final

En observant les deux diagrammes de Gantt, nous pouvons remarquer que le test unitaire nous à pris plus de temps que prévu. En effet, nous avons rencontré plusieurs problèmes durant cette phase du projet. Premièrement, il nous était d'abord impossible de connecter la carte OKDO Nano C100 au réseaux interne à l'établissement. Nous avons faire appel à l'administrateur réseaux de l'IUT afin qu'il puisse régler le problème. Afin qu'il puisse analyser celui-ci, il a dû nous priver de nos cartes durant plusieurs séances, ce qui nous a beaucoup retardé. Finalement, il a réussi à nous connecter au proxy en modifiant les paramètres réseaux de la carte. De plus, nous avons perdu du temps à connecter la caméra Raspberry 4 à la carte Nano C100 Raspberry 4, jusqu'à nous rendre compte ~~que celle-ci était incompatible avec la carte. En~~ effet, la carte OKDO Nano C100 est plutôt rare, et est donc compatible ~~avec peu de dispositifs.~~ Nous avons finalement opté pour réaliser toutes les fonctions secondaires sur la caméra Realsense afin de remédier au problème. Nous avons donc pris un retard de 4 jours pendant les tests unitaires, ce qui a décalé toute la suite du diagramme de Gantt initial. Nous avons malgré tout pu finir le projet dans les temps.

conclusions fausses : c'est un échec d'installation du driver linux de la caméra car il n'est pas intégré à la OKDO nano (ni aux cartes NVidia); par contre la real sense a fonctionné immédiatement contre toute attente !

8 CONCLUSION

Grâce à ce projet, nous avons pris en main le système d'exploitation linux et approfondie nos compétences en Python.
De plus, ce projet nous a demandé une autonomie afin de résoudre les difficultés rencontrées lors du projet. Nous avons atteint l'objectif étant de sonder l'environnement à l'aide d'une caméra et d'en tirer des informations importantes pour la course de l'ENS Paris-Saclay. Cependant nous n'avons pas eu le temps de le donner à temps aux GE2 pour la course, où ils auraient dû ajouter des fonctions stratégiques utilisant nos données récupérées par la caméra.
Nous pouvons toujours améliorer notre système en entrainant par exemple un modèle algorithmique directement sur les voitures conforment à la course ou en trouvant un moyen d'ajouter la carte C100 nano sur la voiture.

9 LISTE DES FIGURES

10 BIBLIOGRAPHIE

https://www.generationrobots.com/fr/205-cameras-de-profondeur

https://fr.rs-online.com/web/p/cameras-de-profondeur/2026352?gb=s

https://www.conrad.fr/fr/p/webcam-full-hd-intel-realsense-depth-camera-d435-1920-x-1080-pixel-1707630.html

Carte OKDO C100 Nano okdo C100 nano :

https://www.okdo.com/fr/getting-started/get-started-with-the-c100-nano-csi-cameras/

Installation camera sur carte OKDO C100 Nano :

Intel® RealSense : https://www.youtube.com/watch?v=lL3zxwN5Lnw

Caméra Luxonis OAK-D Pro : https://docs.luxonis.com/projects/api/en/latest/install/#OKDO C100 Nano

Installation ZED SDK :

+ Vérifier et choisir bonne version https://www.stereolabs.com/developers/release#82af3640d775

+ Installation étape par étape : https://www.stereolabs.com/docs/installation/OKDO C100 Nano

Mettre à jour Jetpack:

https://docs.nvidia.com/OKDO C100 Nano/jetpack/install-jetpack/index.html

Consulter le problème de dpkg quand mettre à jour :
https://forums.developer.nvidia.com/t/solution-dpkg-error-processing-package-nvidia-l4t-bootloader-configure/208627

Consulter le problème de nvidia-container :

https://forums.developer.nvidia.com/t/problem-with-jetpack/241169

Installer Realsense SDK et Realsense Viewer :

Realsense : https://github.com/IntelRealSense/librealsense/blob/master/doc/installation_OKDO C100 Nano.md

OKDO C100 Nano Hacks Nano :

https://github.com/OKDO C100 NanoHacksNano/installLibrealsense.git

Cloner répertoire : https://github.com/IntelRealSense/librealsense.git

Installer la bibliothèque pour accéder les fonctions de caméra stéréoscopie en Python :

https://github.com/IntelRealSense/librealsense/tree/master/wrappers/python

https://github.com/opencv/opencv/wiki/TensorFlow-Object-Detection-API#use-existing-config-file-for-your-model

https://github.com/opencv/opencv/pull/16760

11 ANNEXES

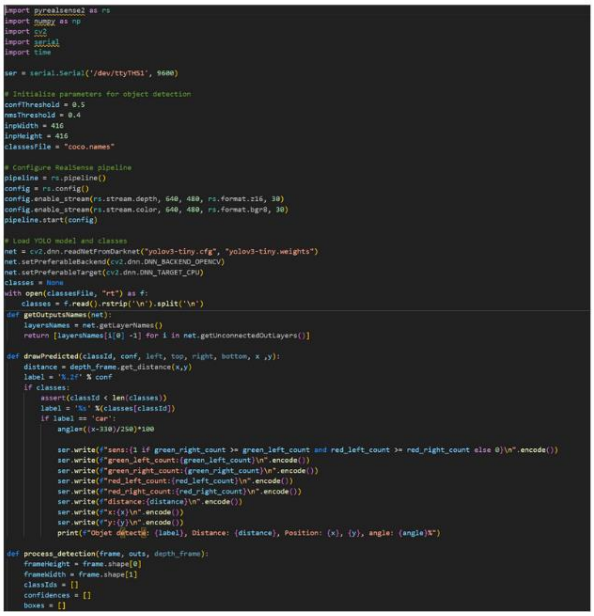
Annexe A : Détection de contours

```
#!/usr/bin/python
# -*- coding utf-8 -*-

import cv2

img = cv2.imread('/home/ti/Images/lena.png', cv2.IMREAD_GRAYSCALE)

import numpy as np

arr = np.array(img)
image = np.zeros((512,512), dtype = np.uint8)

image = cv2.Canny(img, 50, 140)

cv2.imshow('image', img)
cv2.imshow('image ligne detecte', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Annexe B : Détection de couleurs

```python
import cv2
import numpy as np
import pyrealsense2 as rs

pipeline = rs.pipeline()
config = rs.config()
config.enable_stream(rs.stream.color, 640, 480, rs.format.bgr8, 30)
pipeline.start(config)
try:
    while True:
        frames = pipeline.wait_for_frames()
        color_frame = frames.get_color_frame()
        color_image = np.asanyarray(color_frame.get_data())
        gray_image = cv2.cvtColor(color_image, cv2.COLOR_BGR2GRAY)
        blurred_image = cv2.GaussianBlur(gray_image, (5, 5), 0)
        edges = cv2.Canny(blurred_image, 30, 150)

        color_image_with_contours = color_image.copy()
        contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        cv2.drawContours(color_image_with_contours, contours, -1, (0, 255, 0), 2)

        cv2.imshow('Contours Detection', color_image_with_contours)
        cv2.imshow('Original', color_image)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
finally:
    pipeline.stop()
    cv2.destroyAllWindows()
```

Annexe C



```python
import pyrealsense2 as rs
import numpy as np
import cv2
import serial
import time

ser = serial.Serial('/dev/ttyTHS1', 9600)

# Initialize parameters for object detection
confThreshold = 0.5
nmsThreshold = 0.4
inpWidth = 416
inpHeight = 416
classesFile = "coco.names"

# Configure RealSense pipeline
pipeline = rs.pipeline()
config = rs.config()
config.enable_stream(rs.stream.depth, 640, 480, rs.format.z16, 30)
config.enable_stream(rs.stream.color, 640, 480, rs.format.bgr8, 30)
pipeline.start(config)

# Load YOLO model and classes
net = cv2.dnn.readNetFromDarknet("yolov3-tiny.cfg", "yolov3-tiny.weights")
net.setPreferableBackend(cv2.dnn.DNN_BACKEND_OPENCV)
net.setPreferableTarget(cv2.dnn.DNN_TARGET_CPU)
classes = None
with open(classesFile, "rt") as f:
    classes = f.read().rstrip('\n').split('\n')
def getOutputsNames(net):
    layersNames = net.getLayerNames()
    return [layersNames[i[0] -1] for i in net.getUnconnectedOutLayers()]

def drawPredicted(classId, conf, left, top, right, bottom, x ,y):
    distance = depth_frame.get_distance(x,y)
    label = '%.2f' % conf
    if classes:
        assert(classId < len(classes))
        label = '%s' %(classes[classId])
        if label == 'car':
            angle=((x-330)/250)*100

            ser.write(f"sens:{1 if green_right_count >= green_left_count and red_left_count >= red_right_count else 0}\n".encode())
            ser.write(f"green_left_count:{green_left_count}\n".encode())
            ser.write(f"green_right_count:{green_right_count}\n".encode())
            ser.write(f"red_left_count:{red_left_count}\n".encode())
            ser.write(f"red_right_count:{red_right_count}\n".encode())
            ser.write(f"distance:{distance}\n".encode())
            ser.write(f"x:{x}\n".encode())
            ser.write(f"y:{y}\n".encode())
            print(f"Objet détecté: {label}, Distance: {distance}, Position: {x}, {y}, angle: {angle}%")

def process_detection(frame, outs, depth_frame):
    frameHeight = frame.shape[0]
    frameWidth = frame.shape[1]
    classIds = []
    confidences = []
    boxes = []
```

```python
for out in outs:
    for detection in out:
        scores = detection[5:]
        classId = np.argmax(scores)
        confidence = scores[classId]
        if confidence > confThreshold:
            center_x = int(detection[0]*frameWidth)
            center_y = int(detection[1]*frameHeight)
            width = int(detection[2]*frameWidth)
            height = int(detection[3]*frameHeight)
            left = int(center_x - width/2)
            top = int(center_y - height/2)
            classIds.append(classId)
            confidences.append(float(confidence))
            boxes.append([left,top,width,height])
indices = cv2.dnn.NMSBoxes(boxes, confidences, confThreshold, nmsThreshold)
for i in indices:
    i = i[0]
    box = boxes[i]
    left = box[0]
    top = box[1]
    width = box[2]
    height = box[3]
    x = int(left+width/2)
    y = int(top+ height/2)
    drawPredicted(classIds[i], confidences[i], left, top, left+width, top+height, x, y)
if __name__ == "__main__":
    try:
        while True:
            frames = pipeline.wait_for_frames()
            depth_frame = frames.get_depth_frame()
            color_frame = frames.get_color_frame()
            if not depth_frame or not color_frame:
                continue
            color_image = np.asanyarray(color_frame.get_data())
            depth_image = np.asanyarray(depth_frame.get_data())
            blob = cv2.dnn.blobFromImage(color_image, 1/255, (inpWidth, inpHeight), [0,0,0],1,crop=False)
            net.setInput(blob)
            outs = net.forward(getOutputsNames(net))
            process_detection(color_image, outs, depth_frame)

            # Convertir l'image en format HSV pour une meilleure détection des couleurs
            hsv_image = cv2.cvtColor(color_image, cv2.COLOR_BGR2HSV)

            # Définir les plages de couleur pour le vert et le rouge en HSV
            lower_green = (35, 50, 50)
            upper_green = (85, 255, 255)
            lower_red1 = (0, 50, 50)
            upper_red1 = (10, 255, 255)
            lower_red2 = (170, 50, 50)
            upper_red2 = (180, 255, 255)

            # Filtrer les pixels verts et rouges de chaque côté
            mask_green_left = cv2.inRange(hsv_image, lower_green, upper_green)
            mask_green_right = cv2.inRange(hsv_image, lower_green, upper_green)
```

Annexe D :

```
        # Filtrer les pixels verts et rouges de chaque côté
        mask_green_left = cv2.inRange(hsv_image, lower_green, upper_green)
        mask_green_right = cv2.inRange(hsv_image, lower_green, upper_green)
        mask_red1_left = cv2.inRange(hsv_image, lower_red1, upper_red1)
        mask_red1_right = cv2.inRange(hsv_image, lower_red1, upper_red1)
        mask_red2_left = cv2.inRange(hsv_image, lower_red2, upper_red2)
        mask_red2_right = cv2.inRange(hsv_image, lower_red2, upper_red2)

        # Compter les pixels verts et rouges de chaque côté
        green_left_count = cv2.countNonZero(mask_green_left)
        green_right_count = cv2.countNonZero(mask_green_right)
        red_left_count = cv2.countNonZero(mask_red1_left) + cv2.countNonZero(mask_red2_left)
        red_right_count = cv2.countNonZero(mask_red1_right) + cv2.countNonZero(mask_red2_right)

        # Comparer les côtés
        if green_right_count >= green_left_count and red_left_count >= red_right_count:
            sens = True
            print("sens de la course = ", sens)
        else:
            sens = False
            print("sens de la course = ", sens)

        # Afficher les résultats
        print("Nombre de pixels verts - Gauche :", green_left_count, "Droite :", green_right_count)
        print("Nombre de pixels rouges - Gauche :", red_left_count, "Droite :", red_right_count)

        time.sleep(1)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
finally:
    # Arrêter le pipeline RealSense
    pipeline.stop()
    cv2.destroyAllWindows()
```

Annexe E : Détection de couleurs (1)

```
import cv2
import serial
import time

# Initialisation du port série
ser = serial.Serial('/dev/ttyTHS1', baudrate=9600, timeout=1)


# Initialiser la capture vidéo à partir de la webcam
cap = cv2.VideoCapture(0)  # 0 indique la première webcam disponible, vous pouvez changer cela si vous avez plusieurs webcams connectées

if not cap.isOpened():
    print("Erreur: Impossible d'ouvrir la webcam.")
    exit()



cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1920)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 1080)

# Lire continuellement les images de la webcam
while True:
    ret, frame = cap.read()  # Lire une nouvelle image de la webcam
    if not ret:
        print("Erreur: Impossible de lire l'image de la webcam.")
        break

    # Diviser l'image en deux moitiés (gauche et droite)
    height, width, _ = frame.shape
    left_half = frame[:, :width // 2]
    right_half = frame[:, width // 2:]

    # Convertir les images en format HSV pour une meilleure détection des couleurs
    left_hsv = cv2.cvtColor(left_half, cv2.COLOR_BGR2HSV)
    right_hsv = cv2.cvtColor(right_half, cv2.COLOR_BGR2HSV)
```

Annexe F : Détection de couleurs (2)

```
    # Définir les plages de couleur pour le vert et le rouge en HSV
    lower_green = (35, 50, 50)
    upper_green = (85, 255, 255)
    lower_red1 = (0, 50, 50)
    upper_red1 = (10, 255, 255)
    lower_red2 = (170, 50, 50)
    upper_red2 = (180, 255, 255)

    # Filtrer les pixels verts et rouges de chaque côté
    mask_green_left = cv2.inRange(left_hsv, lower_green, upper_green)
    mask_green_right = cv2.inRange(right_hsv, lower_green, upper_green)
    mask_red1_left = cv2.inRange(left_hsv, lower_red1, upper_red1)
    mask_red1_right = cv2.inRange(right_hsv, lower_red1, upper_red1)
    mask_red2_left = cv2.inRange(left_hsv, lower_red2, upper_red2)
    mask_red2_right = cv2.inRange(right_hsv, lower_red2, upper_red2)

    # Compter les pixels verts et rouges de chaque côté
    green_left_count = cv2.countNonZero(mask_green_left)
    green_right_count = cv2.countNonZero(mask_green_right)
    red_left_count = cv2.countNonZero(mask_red1_left) + cv2.countNonZero(mask_red2_left)
    red_right_count = cv2.countNonZero(mask_red1_right) + cv2.countNonZero(mask_red2_right)

    # Comparer les côtés pour déterminer le sens de la course
    if green_right_count >= green_left_count and red_left_count >= red_right_count:
        sens = 1
        print("Sens de la course : ", sens)
    else:
        sens = 0
        print("Sens de la course : ", sens)

    data = f"{sens}, {green_left_count}, {green_right_count}, {red_left_count}, {red_right_count}\n"
    ser.write(data.encode())

```

Annexe G : Détection de couleurs (3)

```
    # Afficher les résultats
    print("Nombre de pixels verts - Gauche :", green_left_count, "Droite :", green_right_count)
    print("Nombre de pixels rouges - Gauche :", red_left_count, "Droite :", red_right_count)

    # Afficher l'image
    cv2.imshow('Frame', frame)

    time.sleep(1)

    # Sortir de la boucle si la touche 'q' est enfoncée
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Libérer la capture vidéo et fermer toutes les fenêtres
cap.release()
cv2.destroyAllWindows()
```

Annexe H

```python
import serial
import time

ser = serial.Serial(port = '/dev/ttyS0', baudrate = 9600, timeout = 1)

try :
    frequency = 10000
    duty_cycle = 50

    while True :
        period = 1 / frequency
        pulse_width = (duty_cycle / 100) * period

        ser.write(b'H')
        time.sleep(pulse_width)
        ser.write(b'L')
        time.sleep(period - pulse_width)

finally :
    ser.close()
```

Annexe I

```python
import serial

# Configuaration du port série
ser  =  serial.Serial('/dev/ttyS0', baudrate = 9600, timeout = 1)

while True :
    # Lire les données du port série
    data = ser.readline().decode().strip()

    # Vérifier si des données ont été reçues
    if data :
        sens = int(data)
        if sens == 1 :
            print("Sens de la course : Avant")
        else :
            print("Sens de la course : Arrière")

ser.close()
```

Annexe J

```python
import serial

# Définir le port série et la vitesse de communication
serial_port = '/dev/ttyS0'  # Mettez ici le port série correct de votre Raspberry Pi
baud_rate = 9600

# Initialiser la communication série
ser = serial.Serial(serial_port, baud_rate)

try:
    data = {}  # Dictionnaire pour stocker les valeurs
    while True:
        # Lire une ligne depuis le port série
        line = ser.readline().decode().strip()

        # Vérifier le préfixe pour déterminer quelle variable est reçue
        if line.startswith('D'):
            data['distance'] = float(line[1:])
        elif line.startswith('X'):
            data['x'] = int(line[1:])
        elif line.startswith('Y'):
            data['y'] = int(line[1:])

        # Vérifier si toutes les valeurs nécessaires ont été reçues
        if len(data) == 3:
            # Afficher les valeurs de distance, x et y en même temps
            print(f"Distance: {data['distance']}, Position X: {data['x']}, Position Y: {data['y']}")
            # Réinitialiser le dictionnaire pour la prochaine série de valeurs
            data = {}

except KeyboardInterrupt:
    print("Arrêt de la réception UART.")
finally:
    ser.close()  # Fermer le port série lorsqu'on quitte le programme
```

Annexe K

```python
import serial
import time

ser = serial.Serial('/dev/ttyS0', 9600)

# Créer un dictionnaire pour stocker les valeurs des variables
variables = {
    "sens": None,
    "green_left_count": None,
    "green_right_count": None,
    "red_left_count": None,
    "red_right_count": None,
    "distance": None,
    "x": None,
    "y": None
}

while True:
    try:
        if ser.in_waiting > 0:
            line = ser.readline().decode().rstrip()
            data = line.split(":")
            if len(data) == 2:
                variable_name = data[0]
                value = data[1]
                # Mettre à jour la valeur de la variable dans le dictionnaire
                if variable_name in variables:
                    variables[variable_name] = value
                # Si toutes les variables ont été mises à jour, imprimer les valeurs
                if all(variables.values()):
                    print("Variables mises à jour :")
                    for var_name, var_value in variables.items():
                        print(f"{var_name}: {var_value}")
                    # Réinitialiser les valeurs des variables dans le dictionnaire
                    variables = {key: None for key in variables.keys()}
    except Exception as e:
        print(f"Erreur : {e}")
        # Attendre un court instant avant de réessayer
        time.sleep(0.1)
```