

PROJET ROBOT SUIVEUR D'ASSISTANCE À LA DÉTECTION DE CHUTES

BENETEAU RYAN

BERNIER RÉMY

DEKKARI YOUNES

GODMUSE THÉO

GRILLET BENOÎT

PREAMBULE

CAHIER DES CHARGES

- La voiture doit être en mesure de s'arrêter à l'approche d'un obstacle (-10cm) ou de s'en écarter
- La voiture doit être en mesure de détecter les personnes, analyser leurs mouvements (chute) et agir en conséquence, le tout à l'aide d'une IA (signal sonore, envoi d'un mail...)
- La voiture doit pouvoir se déplacer de manière autonome dans un environnement inconnu à l'aide de capteurs ultrasons
- La carrosserie devra prévoir des emplacements pour la connectique ainsi que pour le Lidar
- La voiture devra comporter plusieurs capteurs (lidar inclus) lui permettant de comprendre son environnement et naviguer efficacement, en intérieur comme en extérieur
- L'adaptation à son environnement devra se faire à l'aide d'un algorithme en concordance avec les capteurs choisis
- La communication de la chute d'une personne par mail devra se faire de manière sécurisée et centralisée

ETAPE DU PROJET

La voiture :

- Créer une coque pour le véhicule pour lui permettre d'accueillir le sonar, les détecteurs et la caméra
- Configurer les détecteurs
- Configurer la caméra
- Envoyer le programme de reconnaissance de chute dans la carte raspberry pour l'intégrer au véhicule

Le programme :

- Créer un programme pour reconnaître le corps humain
- permettre au programme de détecter une chute du dit humain (si la vitesse de translation horizontal du point de tête est supérieur à 300px/s)
- Réussir à envoyer ce programme dans la raspberry

SOMMAIRE :

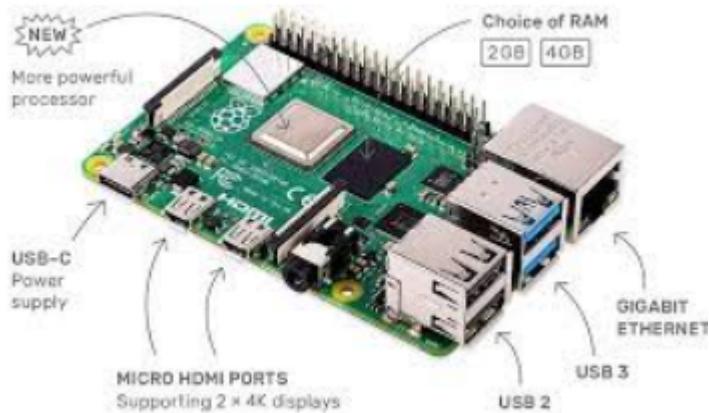
- I – Sujet du projet
- II – Liste de matériel
- III - Equipe
- IV - Organisation
- V – Le véhicule
- VI – La coque
- VII – Programme de détection de personne et de chute
- VIII – Programme de déplacement du véhicule
- IX - Démonstration

SUJET DU PROJET :

Concevoir et déployer un système de contrôle et de navigation autonome exhaustif pour un robot dédié à la patrouille ou à l'accompagnement de groupes de personnes dans des environnements variés, tels que les hôpitaux, maisons de retraite ou les zones publiques. Le robot doit présenter une taille significative en hauteur, visant environ 1,20 mètres, bien qu'une taille réduite puisse être envisagée si nécessaire. Il doit démontrer une autonomie, se déplaçant grâce à une combinaison de lidar et de vision, ou exclusivement par vision, avec la possibilité d'intégrer un système GPS en option pour une navigation précise. L'un des objectifs clés est de détecter les chutes humaines en utilisant la vision et l'intelligence artificielle. La plateforme mobile de développement retenue pour ce projet est le TurtleBot Burger, fournissant une base solide pour la conception et la mise en œuvre de ce robot autonome, polyvalent et conforme aux exigences spécifiées.

LISTE DES ÉQUIPEMENTS :

Matériel	Fournisseur	Prix TTC Indicatif (€)
Turtlebot Burger	GenerationRobots	874.54
1 Chassis		
2 Dynamixel X-series servos		
1 Open Source control board (OpenCR ARM Cortex-M7 with Arduino IDE)		
1 embedded PC (the Raspberry Pi 3 for the Burger version)		
Sensors for navigation (3-axis gyroscope, accelerometer and magnetometer)		
LiDAR LDS-02		
1 battery Li-Po 11.1V 1800mAh		
Camera module for Raspberry Pi (no infrared)	GenerationRobots	24



Travail sur microcontrôleurs, nommément Raspberry et Arduino.

L'ÉQUIPE :



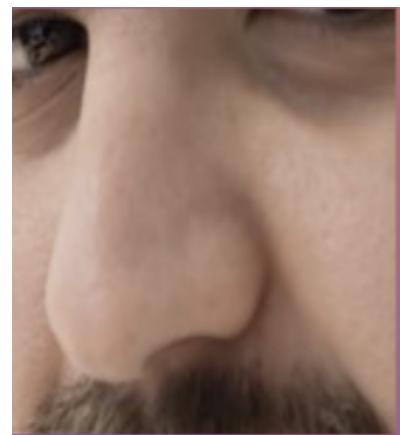
Benoît Grillet

Programmeur Robot



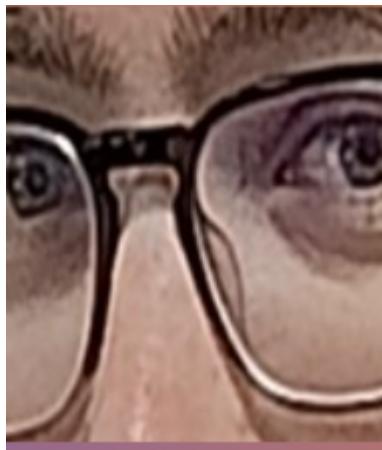
Théo Godmuse

Programmeur IA



Ryan Beneteau

Scrum-Master



Rémy Bernier

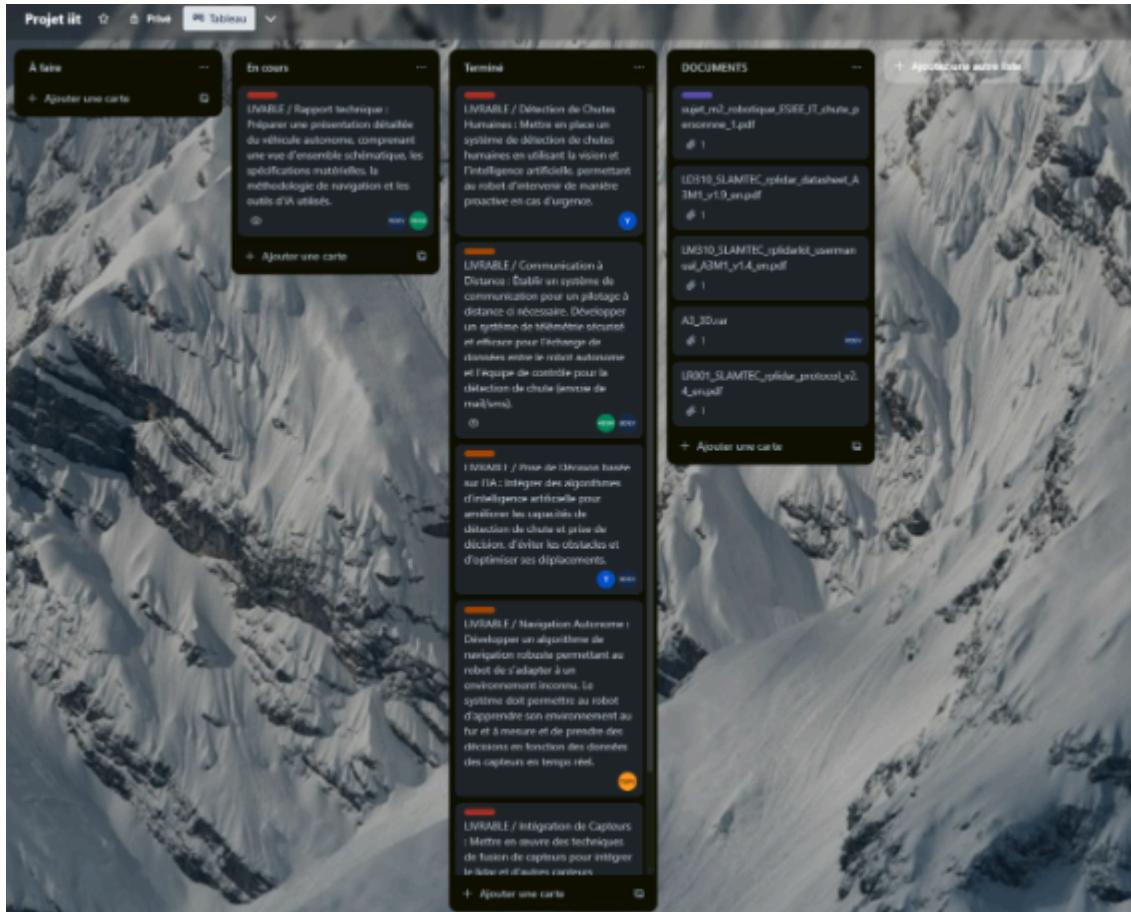
Concepteur Mécanique



Younes Dekkari

Programmeur IA

ORGANISATION :

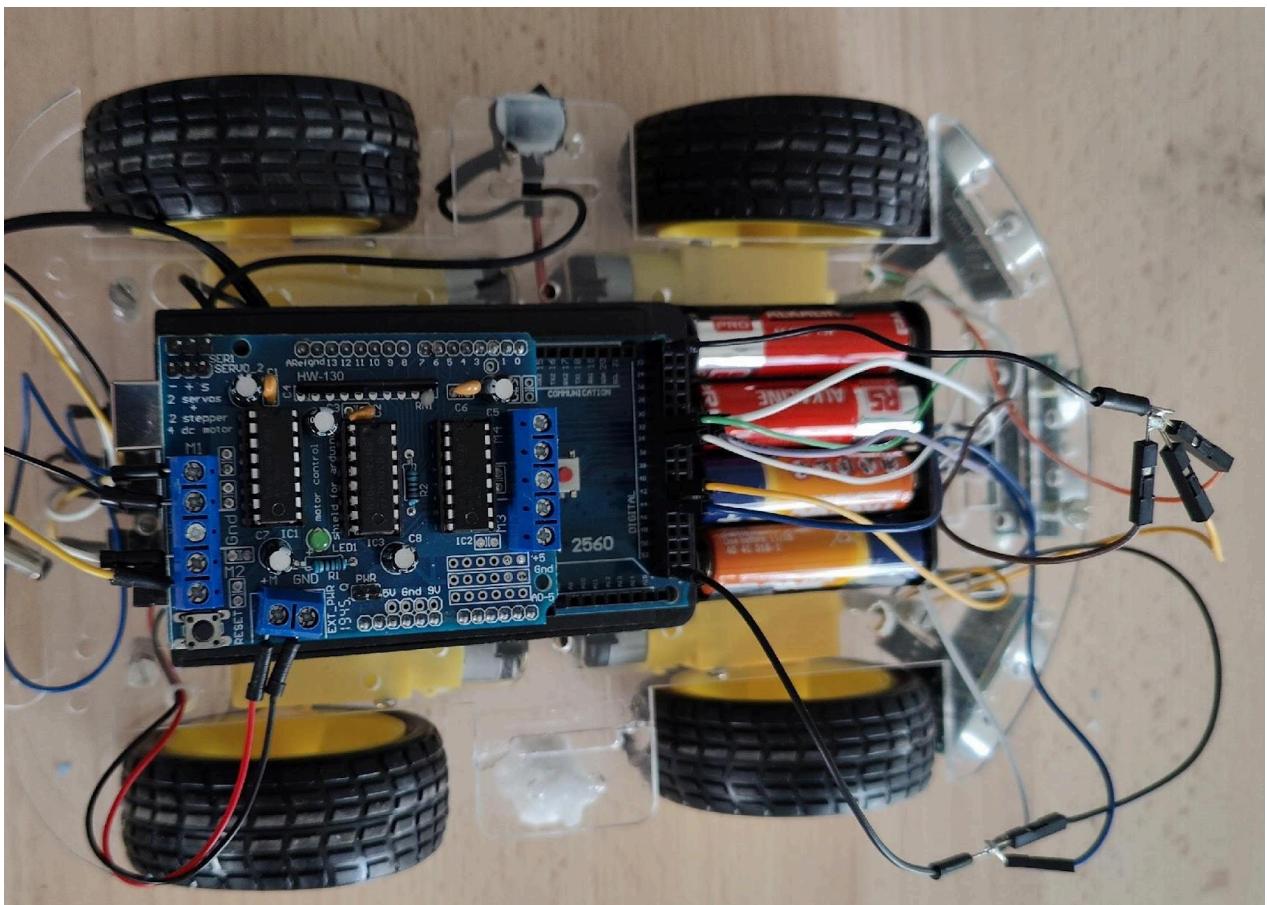
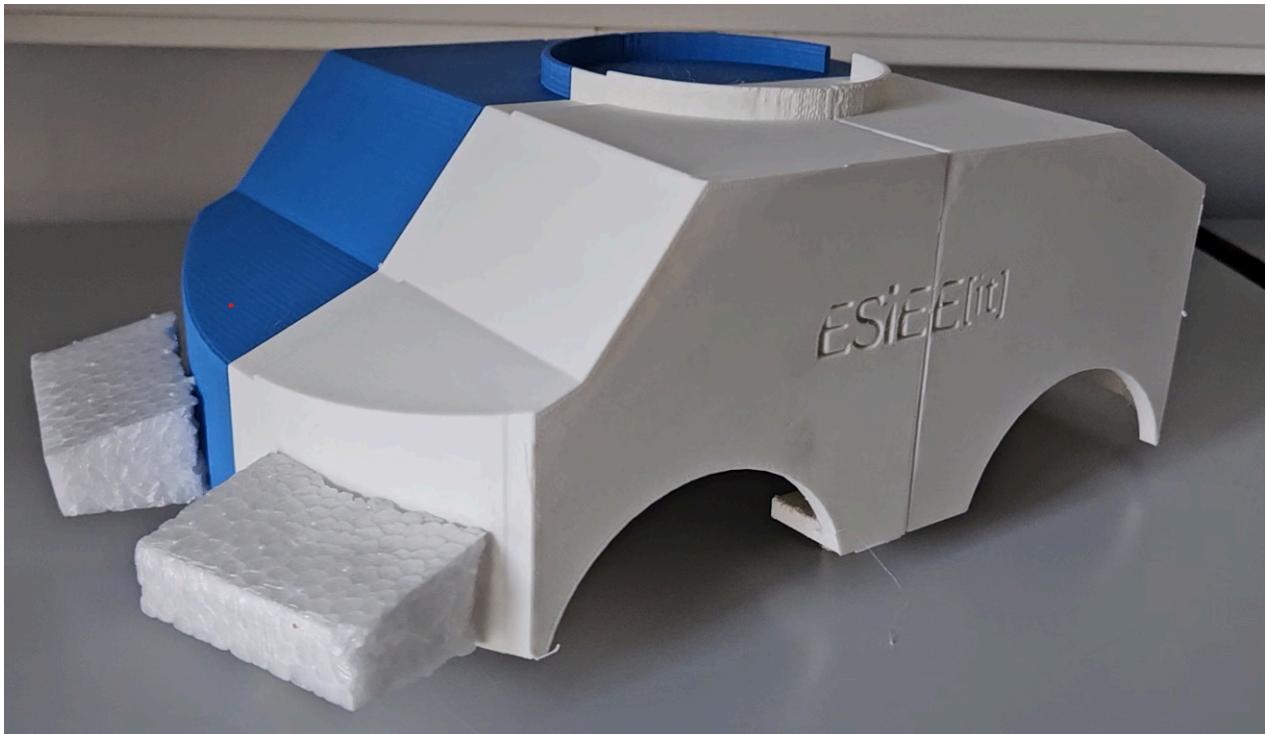


A l'aide du logiciel Trello, nous avons pu dispatcher les ressources de l'équipe équitablement sur chaque tâche, ce qui nous a permis de gérer notre temps de la manière la plus optimale.

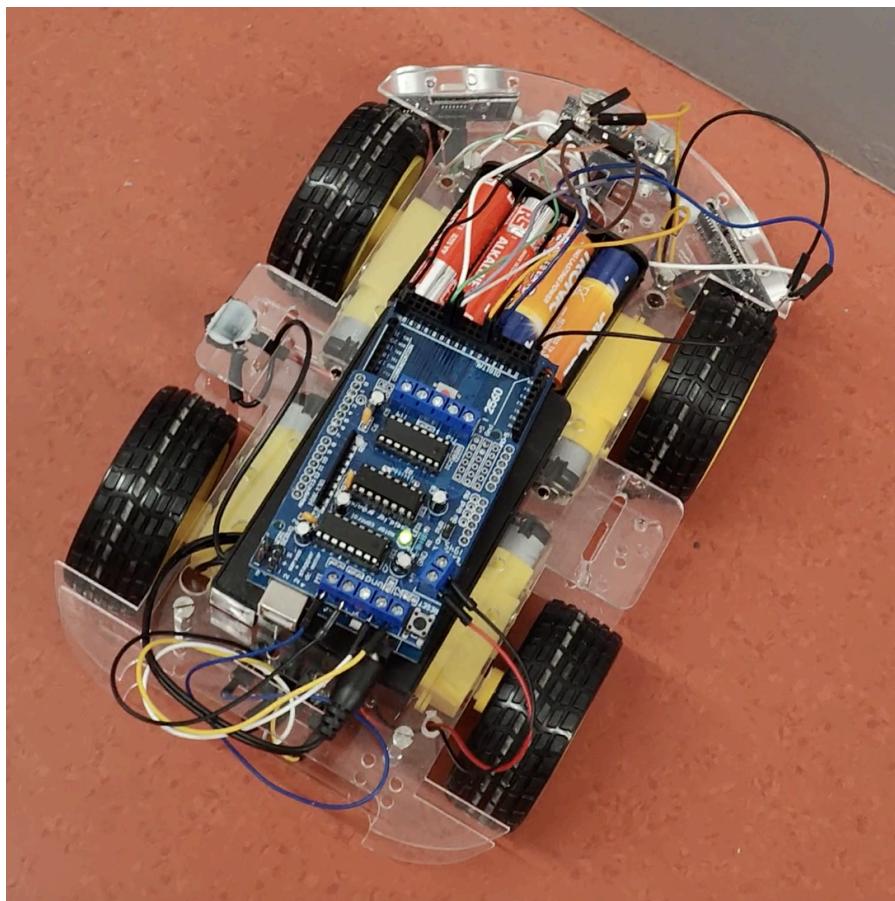
Cela permet d'assurer une traçabilité du projet, et ainsi, voir les problèmes en amont.

Être en mesure de suivre l'avancement du projet, étape par étape nous a permis de redéployer les ressources et les priorités en fonction de notre avancée.

LE VÉHICULE :



COMPOSITION :



- 1 batterie 6V pour alimenter la carte Arduino
- 1 batterie 9V pour alimenter les moteurs
- 1 carte Arduino MEGA 255
- 3 capteurs ultrasons SRF05
- 4 moteurs servo-moteurs

FONCTIONNEMENT DU CODE D'AUTONOMIE:

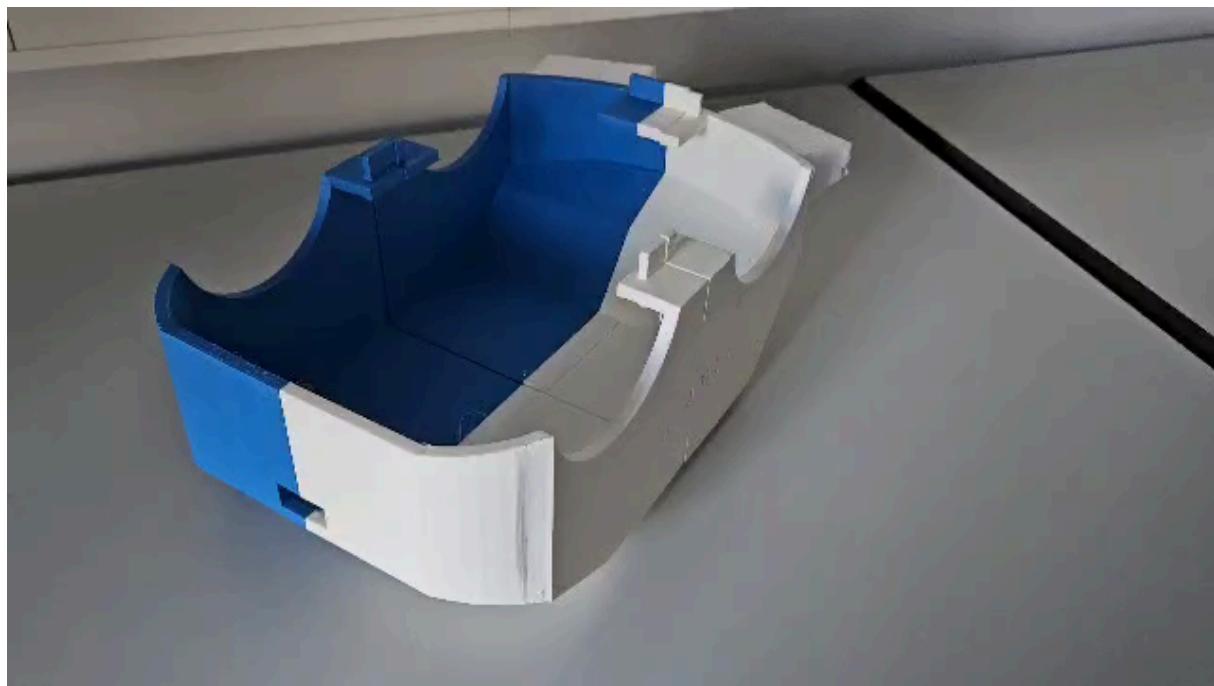


- Détection d'obstacle en continue
- Changement de trajectoire dès que la distance < 20cm



- Problème de vitesse du moteur !
- Problème vitesse de traitement arduino !

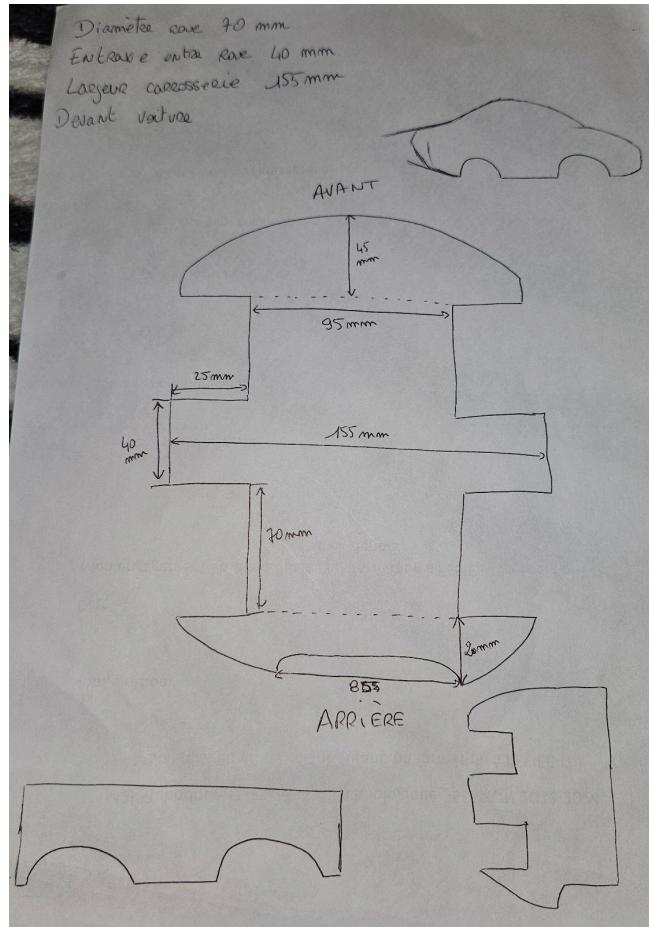
LA COQUE :



1ère ÉBAUCHE :

Dans un premier temps, pour concevoir la coque du véhicule, il a fallu faire quelques schémas pour avoir une première forme. Le schéma que vous pouvez apercevoir à côté représente la forme du châssis de la voiture qui était existant avant la conception de la coque. C'est sur cette pièce que la coque viendra se fixer et se poser.

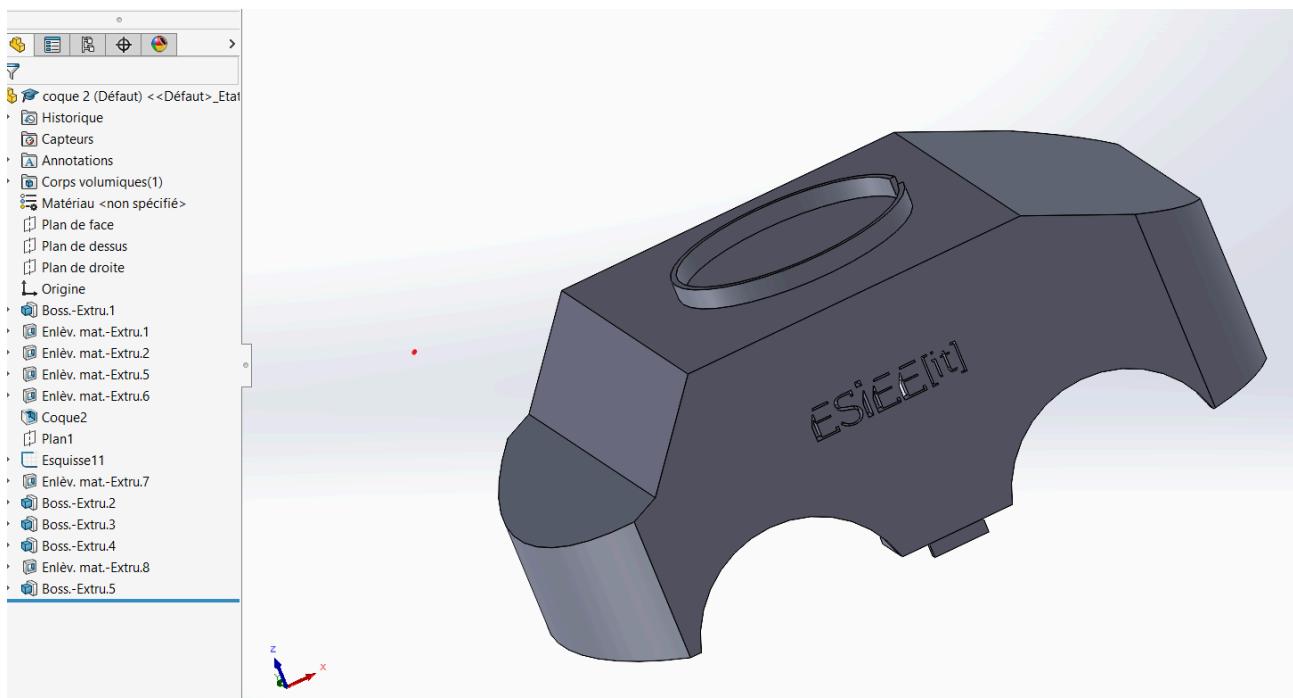
Il a donc fallu prendre toutes les cotations et repérer celles qui étaient importantes. La fixation de la coque sur le châssis s'effectuera par trois petits taquets que l'on montrera plus tard dans la conception (1 à l'avant du véhicule et les 2 autres seront sur les côtés du véhicule entre les roues).



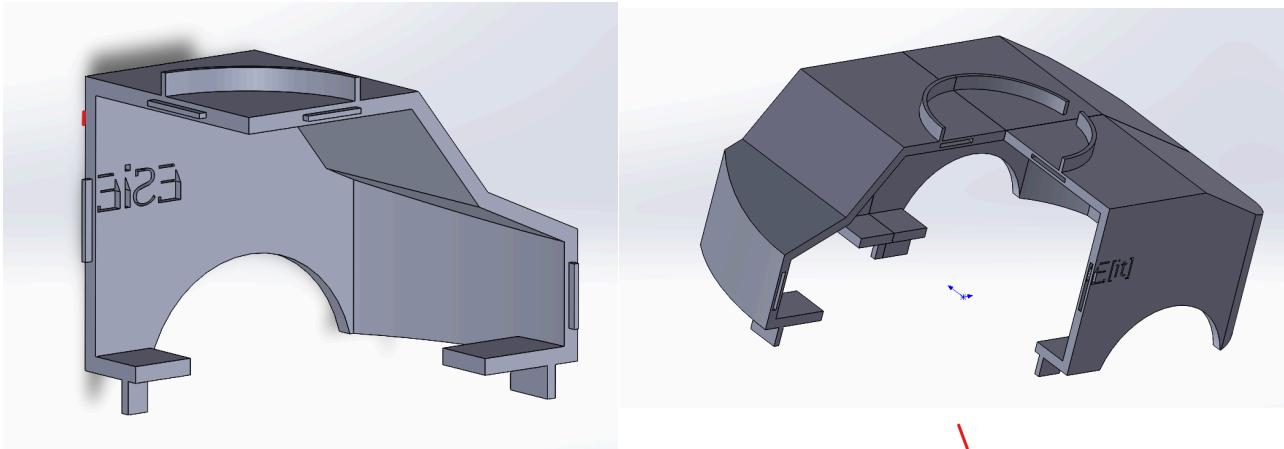
Une fois les premiers schémas réalisés, nous sommes passés à l'étape de conception 3D sur le logiciel SolidWorks.



Pour commencer la conception, nous avons décidé de partir d'un bloc brut et de lui donner la forme que nous avions décidé. Ensuite, le but de la conception était de réaliser une coque pour pouvoir poser sur le châssis et ainsi cacher les composants qui étaient posés dessus. Donc nous avons utilisé la fonction "coque" sur SolidWorks pour vider l'intérieur de notre bloc brut de départ. L'extrusion en forme de cercle sur le dessus permettra de poser le lidar sur la coque de sorte à ce qu'il puisse effectuer ses rotations au-dessus du véhicule. Enfin, sur l'image ci-dessous, nous pouvons apercevoir un taquet qui permettra de fixer la coque sur le châssis.



Une fois la coque réalisée sur SolidWorks, un travail de préparation était à faire pour commencer l'impression des pièces en 3D. La coque étant une grosse pièce, il était impossible de l'imprimer en 1 seule fois. Nous avons donc décidé de découper la coque en 4 parties pour que le temps d'impression soit cohérent et possible.



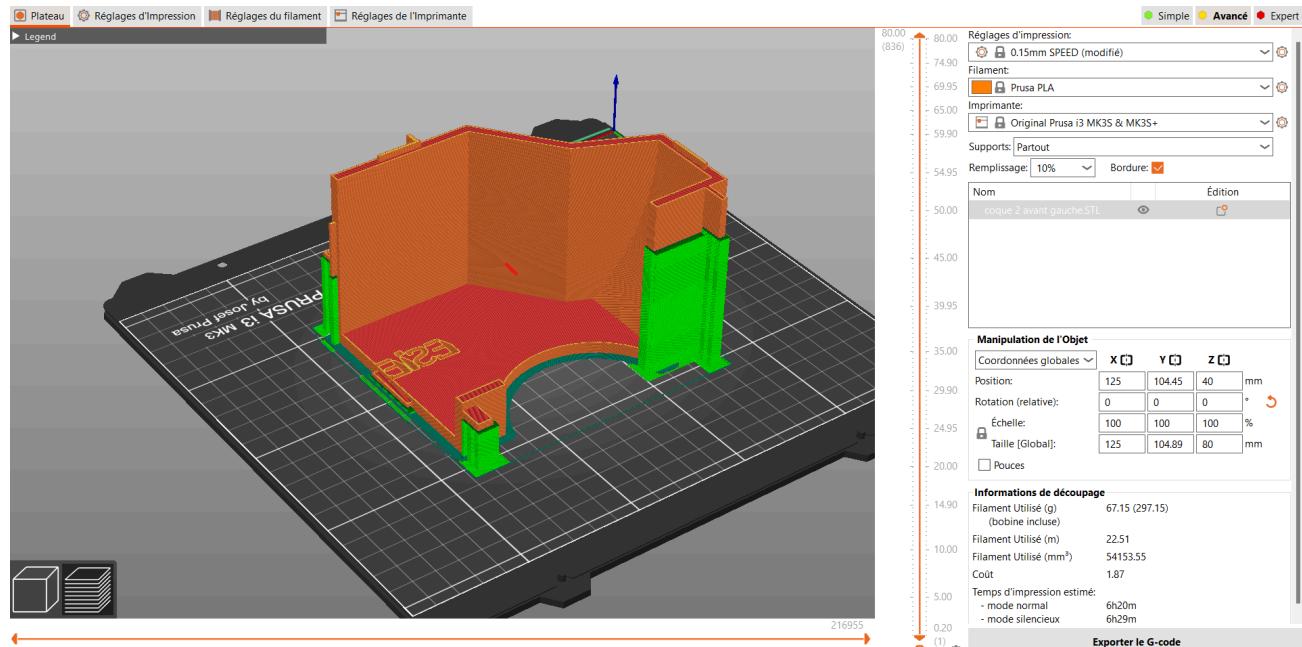
Voici ci-dessus le résultat après la séparation des quatres parties. Nous avons réalisé des encoches et des taquets pour pouvoir emboîter correctement les pièces entre elles une fois l'impression finie.

Maintenant que la conception de la coque est terminée, nous sommes passés à la partie réalisation en 3D sur les imprimantes disponibles au sein de l'école. Les imprimantes de l'école sont des imprimantes Prusa mk3 comme on peut le voir sur la photo ci-dessous.



Pour pouvoir imprimer les pièces, il a fallu installer le logiciel lié au imprimante de la marque Prusa qui se nomme "Prusa Slicer". Pour ouvrir la pièce sur le logiciel d'impression, il faut au préalable enregistrer la pièce sur SolidWorks sous le format "stl".

Sur l'image ci-dessous, Nous pouvons voir une des quatres pièces de notre coque qui est positionnée sur le plateau. Pour une impression optimale, nous avons joué avec quelques paramètres tels que la création de support au niveau des parties qui ne touchent pas le plateau. Le positionnement de la pièce sur le plateau a été choisi en fonction de la forme de la pièce à imprimer, et donc nous avons positionné la pièce sur une face plate.



Voici enfin le résultat de l'impression des quatres pièces qui constituent la coque. Les pièces ont été réalisées en PLA (plastique) et on eu besoin d'environ 6h30 chacune pour être imprimées. Le logo de l'école ESIEE-IT a été dessiné sur la coque pour finaliser notre projet. Des petits blocs de polystyrènes ont été fixé en cas de collisions pour éviter d'abîmer le matériel mis à notre disposition.



Programme de détection de personne et de chute :

```
import cv2 as cv
import argparse
import pygame
from picamera2 import Picamera2

# Initialisation de pygame pour la gestion du son
pygame.mixer.init()

# Variables globales
fall_count = 0
fall_sound_count = 0
prev_head_position = None
prev_timestamp = None
vertical_speed = 0

# Corps du programme
def main():
    global fall_count, fall_sound_count, prev_head_position, prev_timestamp, vertical_speed

    parser = argparse.ArgumentParser()
    parser.add_argument('--thr', default=0.2, type=float, help='Valeur seuil pour la carte de chaleur des parties de la pose')
    parser.add_argument('--width', default=368, type=int, help='Redimensionner l\'entrée à une largeur spécifique.')
    parser.add_argument('--height', default=368, type=int, help='Redimensionner l\'entrée à une hauteur spécifique.')

    args = parser.parse_args()

    # Création des points de référence sur le corps
    BODY_PARTS = {"Nose": 0, "Neck": 1, "RShoulder": 2, "RElbow": 3, "RWrist": 4,
                  "LShoulder": 5, "LElbow": 6, "LWrist": 7, "RHip": 8, "RKnee": 9,
                  "RAngle": 10, "LHip": 11, "LKnee": 12, "LAngle": 13, "REye": 14,
                  "LEye": 15, "REar": 16, "LEar": 17, "Background": 18}

    inWidth = args.width
    inHeight = args.height

    # Chargement du modèle
    net = cv.dnn.readNetFromTensorflow("graph_opt.pb")
```

```
# Initialisation de la caméra Raspberry Pi
try:
    picam2 = Picamera2() # Assurez-vous d'avoir le bon numéro de caméra spécifié
    camera_config = picam2.create_preview_configuration()
    picam2.configure(camera_config)
    picam2.start()
except Exception as e:
    print("Erreur lors de l'initialisation de la caméra:", e)
    exit(1)

# Chargement du son à jouer en cas de chute
fall_sound = pygame.mixer.Sound('sound.mp3') # Assurez-vous d'avoir un fichier
'sound.mp3' dans le même répertoire que votre script

while True:
    try:
        im = picam2.capture_array()
        im = cv.cvtColor(im, cv.COLOR_BGR2RGB) # Convertir l'image en RGB
        im = cv.resize(im, (inWidth, inHeight)) # Redimensionner l'image d'entrée

        frameWidth = im.shape[1]
        frameHeight = im.shape[0]

        # Prétraitement de l'image
        net.setInput(cv.dnn.blobFromImage(im, 1.0, (inWidth, inHeight), (127.5, 127.5, 127.5),
swapRB=True, crop=False))

        # Passage de l'image à travers le réseau
        out = net.forward()
        out = out[:, :19, :, :]

        assert(len(BODY_PARTS) == out.shape[1])

        # Extraction des points de la pose
        points = []
        for i in range(len(BODY_PARTS)):
            heatMap = out[0, i, :, :]
            _, conf, _ = cv.minMaxLoc(heatMap)
            x = (frameWidth * point[0]) / out.shape[3]
            y = (frameHeight * point[1]) / out.shape[2]
            points.append((int(x), int(y)) if conf > args.thr else None)

        # Calcul de la vitesse verticale
        current_head_position = points[BODY_PARTS["Neck"]]
        current_timestamp = cv.getTickCount()
```

```
if current_head_position is not None and prev_head_position is not None and
prev_timestamp is not None:
    # Calcul de la vitesse verticale en pixels par seconde
    vertical_speed = (current_head_position[1] - prev_head_position[1]) /
((current_timestamp - prev_timestamp) / cv.getTickFrequency())

    # Seuil pour la détection de chute
    fall_threshold = 320 # Ajuster selon les besoins

    # Détection de chute
    if vertical_speed > fall_threshold:
        fall_count += 1
        print("Chute détectée!")

    while fall_sound_count < 4:

        # Jouer le son en cas de chute
        fall_sound.play()
        fall_sound_count += 1
        pygame.time.delay(1000)

    fall_sound_count = 0

    # Mise à jour des variables pour la prochaine itération
    prev_head_position = current_head_position
    prev_timestamp = current_timestamp

    # Affichage du nombre de chutes en permanence à l'écran
    cv.putText(im, f'Nombre de Chutes: {fall_count}', (10, 50),
cv.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)

    # Affichage de la vitesse verticale
    cv.putText(im, f'Vitesse Verticale: {vertical_speed:.2f} pixels/s', (10, 80),
cv.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

    # Affichage des schémas de détection des parties du corps
    for pair in [[1, 2], [1, 5], [2, 3], [3, 4], [5, 6], [6, 7], [1, 8], [8, 9], [9, 10], [1, 11], [11, 12],
[12, 13], [1, 0], [0, 14], [14, 16], [0, 15], [15, 17]]:
        partFrom = pair[0]
        partTo = pair[1]

        if points[partFrom] and points[partTo]:
            cv.line(im, points[partFrom], points[partTo], (0, 255, 0), 3)
            cv.ellipse(im, points[partFrom], (3, 3), 0, 0, 360, (0, 0, 255), cv.FILLED)
            cv.ellipse(im, points[partTo], (3, 3), 0, 0, 360, (0, 0, 255), cv.FILLED)

cv.imshow('OpenPose using OpenCV', im)
```

```
# Sortir de la boucle lorsque la touche 'q' est pressée
if cv.waitKey(1) & 0xFF == ord('q'):
    break

except KeyboardInterrupt:
    break

# Arrêt de la caméra et libération des ressources
picam2.stop()
cv.destroyAllWindows()

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

Programme de déplacement du véhicule :

```
#include "AFMotor.h"

AF_DCMotor motor1(1); // création de l'objet "motor1" - moteurs du côté droit
AF_DCMotor motor2(2); // création de l'objet "motor2" - moteurs du côté gauche

const int EchoR_PIN = 34;
const int EchoM_PIN = 35;
const int EchoL_PIN = 44;
const int TrigR_PIN = 32;
const int TrigM_PIN = 33;
const int TrigL_PIN = 42;

const int VITESSE = 225;
const int SEUIL_DISTANCE = 20;
const int SEUIL_DISTANCE_L = 20;

float distanceR;
float distanceL;
float distanceM;

void release() {
    motor1.run(RELEASE);
    motor2.run(RELEASE);
}

void brake() {
    motor1.run(BRAKE);
    motor2.run(BRAKE);
}

void forward() {
    release();
    motor1.run(FORWARD);
    motor2.run(FORWARD);
    motor1.setSpeed(VITESSE); //max 255
    motor2.setSpeed(VITESSE);
}
```

```
}

void backward() {
    release();
    motor1.run(BACKWARD); //sens du moteur
    motor2.run(BACKWARD);
    motor1.setSpeed(VITESSE); //choix de la vitesse du moteur (max 255)
    motor2.setSpeed(VITESSE);
}

void turnLeft(){ //un coté en marche arrière et l'autre à l'arrêt
    release();
    motor1.run(BACKWARD); //marche arriere
    motor2.run(BRAKE); //arret
    motor1.setSpeed(VITESSE); //choix de la vitesse du moteur (max 255)
}

void turnRight(){
    release();
    motor1.run(BRAKE);
    motor2.run(BACKWARD);
    motor2.setSpeed(VITESSE);
}

void getSignal(int triggerPin){ //récupération de la valeur des capteurs ultrasons
    digitalWrite(triggerPin, LOW);
    delayMicroseconds(2);
    digitalWrite(triggerPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(triggerPin, LOW);
}

void setup() {
    pinMode(EchoM_PIN, INPUT); //déclaration des in out
    pinMode(TrigM_PIN, OUTPUT);
    pinMode(EchoR_PIN, INPUT);
    pinMode(TrigR_PIN, OUTPUT);
    pinMode(EchoL_PIN, INPUT);
    pinMode(TrigL_PIN, OUTPUT);
}
```

```
}
```

```
void loop() {
```

```
    getSignal(TrigR_PIN); //appel de la fonction ultrasons
```

```
    distanceR = (pulseIn(EchoR_PIN, HIGH))/29.0/2.0; //conversion en centimètre
```

```
    getSignal(TrigL_PIN);
```

```
    distanceL = (pulseIn(EchoL_PIN, HIGH))/29.0/2.0;
```

```
    getSignal(TrigM_PIN);
```

```
    distanceM = (pulseIn(EchoM_PIN, HIGH))/29.0/2.0;
```

```
    //Comparaison des distances des 3 capteurs pour les différents cas possible
```

```
    if(distanceM < SEUIL_DISTANCE) {
```

```
        if(distanceL < distanceR) {
```

```
            turnRight();
```

```
        } else {
```

```
            turnLeft();
```

```
        }
```

```
    }
```

```
    if((distanceR < SEUIL_DISTANCE) && (distanceL < SEUIL_DISTANCE_L) && (distanceM < SEUIL_DISTANCE)) { //arrêt
```

```
        release();
```

```
    }
```

```
    if((distanceR < SEUIL_DISTANCE) && (distanceL > SEUIL_DISTANCE_L)) { //tourner à gauche
```

```
        turnLeft();
```

```
    }
```

```
    if((distanceL < SEUIL_DISTANCE_L) && (distanceR > SEUIL_DISTANCE)) { //tourner à droite
```

```
        turnRight();
```

```
    }
```

```
    if((distanceR > SEUIL_DISTANCE) && (distanceL > SEUIL_DISTANCE_L) && (distanceM > SEUIL_DISTANCE)) { //avancer
```

```
        forward();
```

```
    }
```

```
}
```

Programme de la raspberry pi :

```
import cv2 as cv

import argparse

import pygame

from picamera2 import Picamera2


# Initialisation de pygame pour la gestion du son

pygame.mixer.init()


# Variables globales

fall_count = 0

fall_sound_count = 0

prev_head_position = None

prev_timestamp = None

vertical_speed = 0


# Corps du programme

def main():

    global fall_count, fall_sound_count, prev_head_position,
prev_timestamp, vertical_speed


    parser = argparse.ArgumentParser()

    parser.add_argument('--thr', default=0.2, type=float, help='Valeur
seuil pour la carte de chaleur des parties de la pose')

    parser.add_argument('--width', default=368, type=int,
help='Redimensionner l\'entrée à une largeur spécifique.')

    parser.add_argument('--height', default=368, type=int,
help='Redimensionner l\'entrée à une hauteur spécifique.')

    args = parser.parse_args()
```

```
BODY_PARTS = {"Nose": 0, "Neck": 1, "RShoulder": 2, "RElbow": 3,
"RWrist": 4,
"LShoulder": 5, "LElbow": 6, "LWrist": 7, "RHip": 8,
"RKnee": 9,
"RAngle": 10, "LHip": 11, "LKnee": 12, "LAngle": 13,
"REye": 14,
"LEye": 15, "REar": 16, "LEar": 17, "Background": 18}

inWidth = args.width
inHeight = args.height

# Chargement du modèle
net = cv.dnn.readNetFromTensorflow("graph_opt.pb")

# Initialisation de la caméra Raspberry Pi
try:
    picam2 = Picamera2() # Assurez-vous d'avoir le bon numéro de
caméra spécifié
    camera_config = picam2.create_preview_configuration()
    picam2.configure(camera_config)
    picam2.start()
except Exception as e:
    print("Erreur lors de l'initialisation de la caméra:", e)
    exit(1)

# Chargement du son à jouer en cas de chute
fall_sound = pygame.mixer.Sound('sound.mp3') # Assurez-vous
d'avoir un fichier 'sound.mp3' dans le même répertoire que votre script

while True:
```

```
try:

    im = picam2.capture_array()

    im = cv.cvtColor(im, cv.COLOR_BGR2RGB)    # Convertir l'image
en RGB

    im = cv.resize(im, (inWidth, inHeight))    # Redimensionner
l'image d'entrée


    frameWidth = im.shape[1]
    frameHeight = im.shape[0]

    # Prétraitement de l'image

    net.setInput(cv.dnn.blobFromImage(im, 1.0, (inWidth,
inHeight), (127.5, 127.5, 127.5), swapRB=True, crop=False))

    # Passage de l'image à travers le réseau

    out = net.forward()

    out = out[:, :19, :, :]

    assert(len(BODY_PARTS) == out.shape[1])

    # Extraction des points de la pose

    points = []

    for i in range(len(BODY_PARTS)):

        heatMap = out[0, i, :, :]

        _, conf, _, point = cv.minMaxLoc(heatMap)

        x = (frameWidth * point[0]) / out.shape[3]

        y = (frameHeight * point[1]) / out.shape[2]

        points.append((int(x), int(y)) if conf > args.thr else
None)
```

```
# Calcul de la vitesse verticale

current_head_position = points[BODY_PARTS["Neck"]]

current_timestamp = cv.getTickCount()

if current_head_position is not None and prev_head_position is not None and prev_timestamp is not None:

    # Calcul de la vitesse verticale en pixels par seconde

    vertical_speed = (current_head_position[1] - prev_head_position[1]) / ((current_timestamp - prev_timestamp) / cv.getTickFrequency())

    # Seuil pour la détection de chute

    fall_threshold = 320

    # Détection de chute

    if vertical_speed > fall_threshold:

        fall_count += 1

        print("Chute détectée!")

while fall_sound_count < 4:

    # Jouer le son en cas de chute

    fall_sound.play()

    fall_sound_count += 1

    pygame.time.delay(1000)

    fall_sound_count = 0

# Mise à jour des variables pour la prochaine itération

prev_head_position = current_head_position
```

```
prev_timestamp = current_timestamp

# Affichage du nombre de chutes en permanence à l'écran

cv.putText(im, f'Nombre de Chutes: {fall_count}', (10, 50),
cv.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)

# Affichage de la vitesse verticale

cv.putText(im, f'Vitesse Verticale: {vertical_speed:.2f} pixels/s', (10, 80), cv.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

# Affichage des schémas de détection des parties du corps

for pair in [[1, 2], [1, 5], [2, 3], [3, 4], [5, 6], [6, 7], [1, 8], [8, 9], [9, 10], [1, 11], [11, 12], [12, 13], [1, 0], [0, 14], [14, 16], [0, 15], [15, 17]]:

    partFrom = pair[0]
    partTo = pair[1]

    if points[partFrom] and points[partTo]:
        cv.line(im, points[partFrom], points[partTo], (0, 255, 0), 3)

        cv.ellipse(im, points[partFrom], (3, 3), 0, 0, 360,
(0, 0, 255), cv.FILLED)

        cv.ellipse(im, points[partTo], (3, 3), 0, 0, 360,
(0, 0, 255), cv.FILLED)

    cv.imshow('OpenPose using OpenCV', im)

# Sortir de la boucle lorsque la touche 'q' est pressée

if cv.waitKey(1) & 0xFF == ord('q'):

    break
```

```
except KeyboardInterrupt:  
  
    break  
  
  
# Arrêt de la caméra et libération des ressources  
picam2.stop()  
  
cv.destroyAllWindows()  
  
  
if __name__ == "__main__":  
    main()
```

Etapes :

CAPTURER L'IMAGE AVEC
PICAMERA2

Convertir l'image en RGB

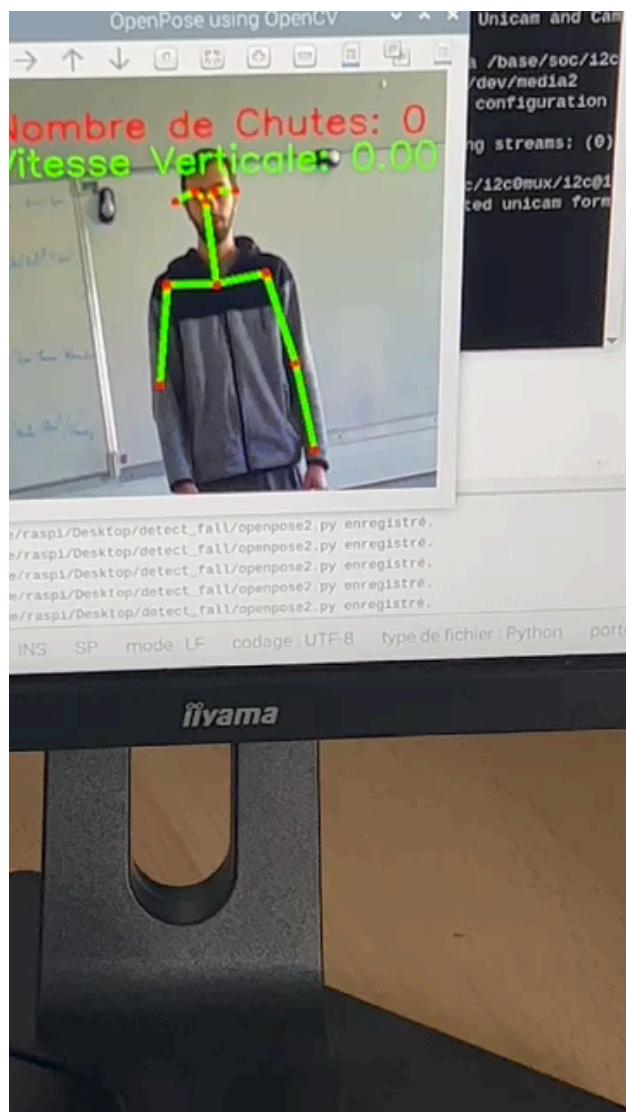
Régler la résolution

Prétraitement de l'image

Passage de l'image en réseau:

- Responsables du passage de l'image prétraitée à travers le réseau de neurones pour obtenir les prédictions de pose

Démonstration :



+ Voir vidéo de présentation