POLYTECH LILLE

# Rapport de Projet

Sujet : Robot ramasseur de déchet

Présenté par : DEFFRENNES Vianney - DELANNOY Jason - THOMAS Gabriel

# Sommaire

# Introduction

Ce dossier est le rapport de projet du groupe de Vianney DEFFRENNES, Jason DELANNOY et Gabriel THOMAS. Il intervient dans le module projet du Semestre 8 de Systèmes Embarquées.

Le projet consiste en la réalisation d'un système mobile et autonome permettant la reconnaissance et la saisie de mégots de cigarettes au sol. Ce projet est encadré par M. Othman Lakhal. Nous avons décidé de nommer notre robot "Trashy" (qui vient de déchet en anglais)

Ce document va compiler tout notre travail sur les trois derniers semestres de la spécification du cahier des charges à la réalisation d'une preuve de concept.

# I. Semestre 6

Lors de ce semestre, nous avons pris en main le projet et nous avons défini le cahier des charges techniques et fonctionnelles de notre projet.

## A. Analyse des besoins

Pour notre étude, nous avons d'abord établi une analyse des besoins de notre projet. Après recherche, nous avons découvert, qu'en France, près de 68 Milliards de cigarettes étaient fumés par an. Et parmi ces 68 Milliards, on estime à 40 Milliards de mégots jetés au sol.

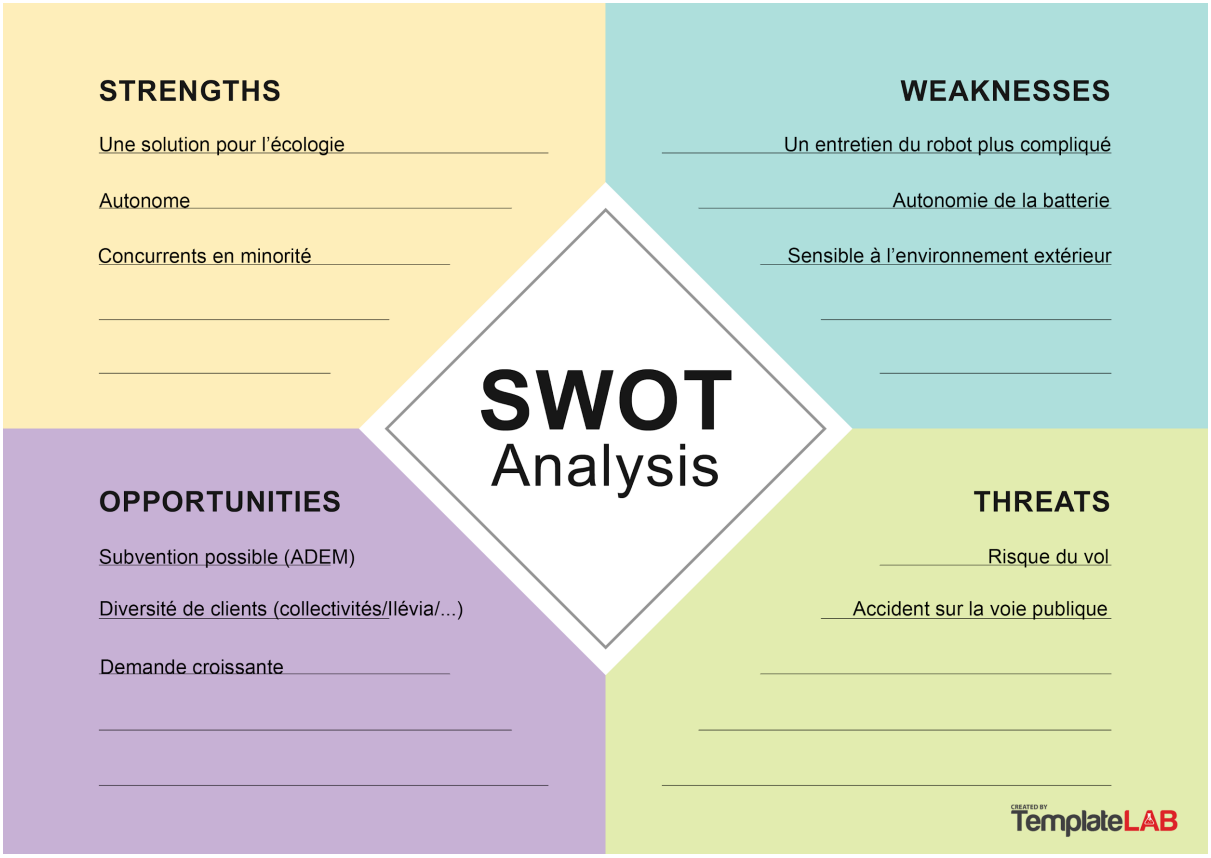
Comme un exemple sera beaucoup plus parlant, prenons l'exemple d'un défi réalisé par 230 participants aux Champs-Elysées. Pour sensibiliser sur le nombre de mégots jetés aux Champs-Elysées, 230 bénévoles se sont mis pour défis de ramasser pendant 1h30 tous les mégots qu'ils trouvaient sur leurs chemins. Une fois ramassés, ils ont compté un total d'environ 100000 mégots. Et ça, ce n'est qu'à l'échelle des Champs-Elysées. A partir de là, on peut s'imaginer la quantité de mégots qui se trouve actuellement sur le sol de nos villes.

Nous avons donc pour projet de créer un robot autonome capable de se déplacer vers le mégot et de le ramasser en le disposant dans un récipient.

## B. Etude d'opportunité

Comme vu précédemment, nous voulons une solution à ce problème en utilisant toutes les ressources dont nous disposons.

Nous avons donc, avec l'aide de notre professeur d'Economie du semestre 5 Mr Joël Ferry, réalisé un diagramme SWOT afin de mettre au point les forces et faiblesses du projet et ainsi mieux s'orienter :

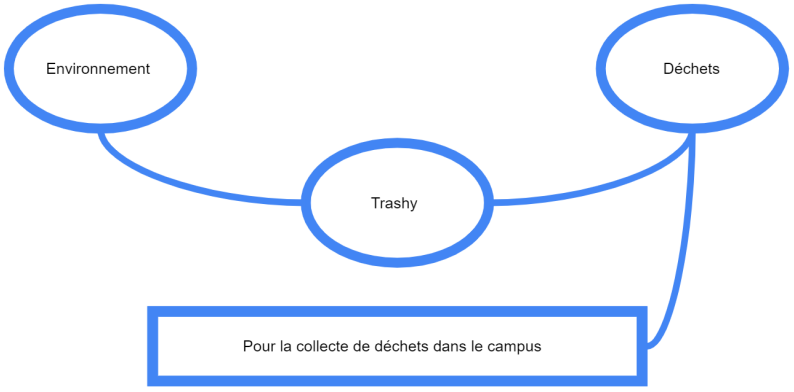



Suite à l'entretien avec notre professeur d'économie, nous avions décidé de partir sur un robot ramasseur de déchets et ainsi le rendre plus polyvalent. Après discussion avec le reste du groupe, la plupart des menaces évoquées peuvent être soit évitées, soit réparées en ajoutant un module GPS pour retrouver le robot en cas de vol par exemple.

Pour ce projet et comme vu dans l'analyse SWOT, il y a une diversité de clients comme les collectivités locales, les sociétés de transports tel qu'Ilevia et la SNCF (pour le nettoyage en gare ou station). On peut aussi envisager toutes les subventions de l'ADEM. Nous aimerions aussi rendre la technologie efficace pour pouvoir diminuer les coûts de fabrication.

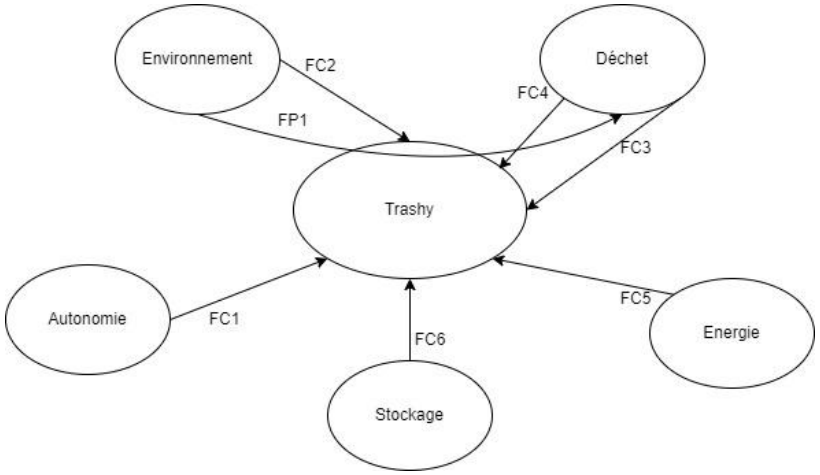
## C. Définition du cahier des charges

Pour définir notre cahier des charges, nous avons établie les fonctions principales du robot dans ce diagramme bête à corne :




Ainsi, nous avons établi que le robot évolue dans un environnement extérieur (initialement le campus mais pouvant aller plus loin) et permettra la collecte des mégots.

Nous avons ensuite étoffé les fonctions du robot dans un diagramme des interacteurs ci-dessous :




| | |
|---|---|
| FP1 | Ramasser les déchets |
| FC1 | Le robot doit pouvoir se mouvoir de façon autonome |
| FC2 | Le robot doit être capable de rouler sur tout type de terrain |
| FC3 | Le robot doit être apte à repérer les déchets |
| FC4 | Le robot doit pouvoir récupérer les déchets |
| FC5 | Avoir suffisamment d'énergie |
| FC6 | Stocker les déchets |

Nous avons donc déterminé les fonctions contraintes du robot qu'il doit pouvoir réaliser telle la détection des mégots ou encore la contrainte de devoir évoluer sur tout type de terrain. Et c'est ainsi, et après concertation avec tous les membres du projets, que nous avons écrit le cahier des charges suivants :

- Le robot devra rouler à une vitesse de 4km/h et devra être capable de passer sur des obstacles de 2cm maximum. De plus, lorsque le robot détecte un obstacle qu'il ne peut peut franchir, il doit s'arrêter et le contourner. Enfin, en cas d'obstacle dangereux, le robot devra émettre un son.
- Lorsque le robot détecte un déchet, il doit l'attraper à l'aide de sa pince puis le mettre dans son réservoir. Si il est plein, ou si la masse maximale que le robot peut transporter est atteinte, le robot devra partir vers une station de déchargement (faite par le client, hors du cadre de notre projet) pour se vider. De plus, lorsque le robot ramasse un déchet, il devra cartographier ce déchet afin de connaître les zones les plus polluées. A noter que le robot ne fait pas le tri des déchets.
- La position du robot doit être connue en temps réel afin d'analyser les données du parcours et le robot devra connaître son orientation à l'aide de son gyroscope.
- Le robot doit avoir une autonomie de  8h et doit pouvoir repartir vers la station de charge si sa batterie devient faible.
- Les bras du robot sont en caoutchouc pour plus d'adhérence et sa trappe (de 41L) est concave afin d'éviter que les déchets ne se coincent.
- Le robot doit être capable d'évoluer dans des terrains semi-accidentés et de monter un trottoir. En cas d'urgence, le robot doit immédiatement s'arrêter.

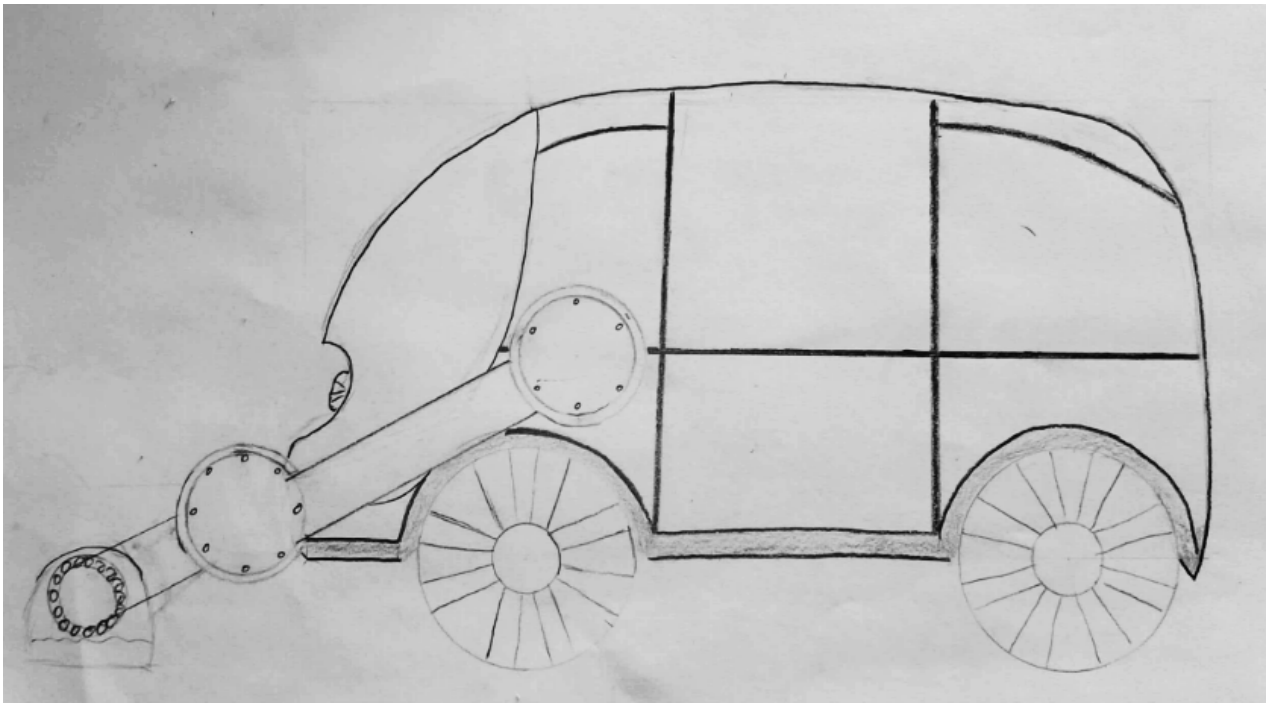
## D. La solution retenue

Pour répondre à notre problématique, nous allons réaliser un robot tout-terrain permettant de détecter et ramasser les déchets. Nous avons exploré différentes solutions et design pour le robot :

- Tout d'abord, nous avons pensé à partir sur une base de robot aspirateur mais l'adapter pour un environnement extérieur.
- Comme la solution précédente, nous avons aussi pensé à partir sur une base de Robotino mais l'adapter pour un environnement extérieur
- Nous avons ensuite pensé à miniaturiser les véhicules de nettoyage de la voie publique

C'est ainsi qu'après réflexions et brainstorming entre les membres du groupe, nous avons trouvé les premières solutions et design de Trashy. Ainsi, nous allons concevoir une carte mère adaptée au robot permettant toutes les fonctionnalités décrites ci-dessus. Nous allons donc avoir recours à un processeur ATMEGA 2560 en communication avec une Raspberry PI. La Raspberry est là pour les traitements des capteurs plus gourmands (dont l'analyse d'image de la caméra) que l'arduino ne peut faire.

Pour le design du robot, nous sommes venu à cette conclusion ci-dessous :



# II. Semestre 7

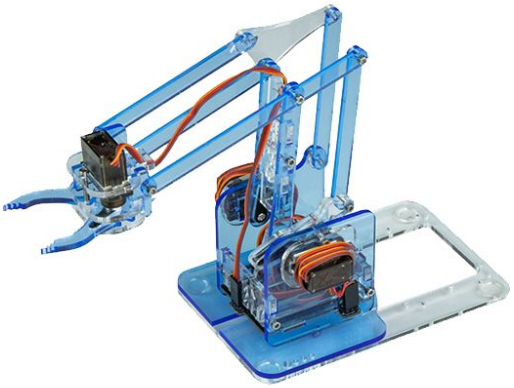
Lors de ce semestre, nous avons commencé la réalisation de notre preuve de concept en nous concentrant sur la détection et le ramassage de mégots de cigarette. Nous avons séparé le projet en plusieurs étapes : le ramassage d'objets, le déplacement, la gestion des déchets et la détection des déchets.

## A. Contrôle de la pince

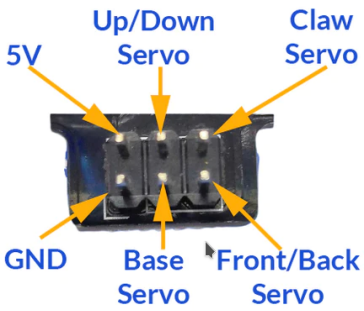
Premièrement, nous avons utilisé la pince MeArm 1.0 que nous avons relié à une carte Arduino

Pour que le robot puisse être capable de prendre les mégots, nous avons dû coder un algorithme en Arduino pour que la pince puisse avancer d'avant en arrière, tourner sur sa base, ouvrir et fermer la pince. Nous avons utilisé la librairie arduino servo.h pour nous aider. Nous avons également trouvé les numéros des pins de la pince correspondant sur internet

Remarque : le délai de 15 ms dans les fonctions sert à contrôler plus doucement les servos moteurs et éviter de les endommager, de même que créer un tableau qui s'incrémente pour éviter les sauts de positions



*Pince MeArm 1.0*




*Port des Servomoteurs de la pince*

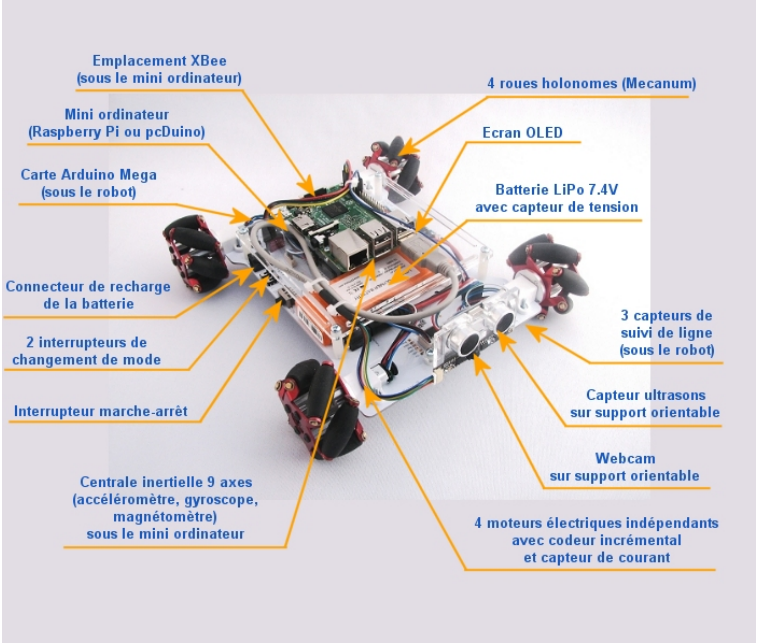
Nous développons ainsi un code générique pour contrôler les différents servomoteurs de la pince ainsi que les mouvements pour attraper un objet :

```
void to_move(Servo obj, int start_angle, int stop_angle){
  /* Ce programme permet de contrôler les servomoteurs de la pince de
Trashy
   *
   * Liste des paramètres :
   * :Servo obj: Servomoteur à actionner
   * :int start_angle: Angle de départ
   * :int stop_angle: Angle d'arrivée
   *
   * CU : Avoir les servomoteurs branché aux bons ports
   *  claw prend des angles de 0 à 95°
   *  Les autres de 0 à 180°
   *
   * Exemple
```

```
   *  to_move(claw,0,95); pour fermer la pince
   *  to_move(claw,95,0); pour ouvrir la pince
   *  to_move(front_back,180,0); pour reculer la pince
   *  to_move(front_back,0,180); pour avancer la pince
   *  to_move(up_down,180,0); pour baisser la pince
   *  to_move(up_down,0,180); pour monter la pince
   *  to_move(base,0,180); pour emmener la pince vers le depos
   *  to_move(base,180,0); pour ramener la pince devant
   */
  if(start_angle < stop_angle){
    for(angle = start_angle; angle < stop_angle; angle += 1){
      obj.write(angle);
      delay(step_time);
    }
  }else{
    for(angle = start_angle; angle >= stop_angle; angle -= 1){
      obj.write(angle);
      delay(step_time);
    }
  }
  delay(100);
}


void attrape_object(){
  to_move(claw,95,0);
  delay(1000);
  to_move(front_back,0,180);
  delay(1000);
  to_move(claw,0,95);
  delay(1000);
  to_move(front_back,180,0);
  delay(1000);
}
```

## B. Mobilité du Robot

Le robot que nous avons reçu est un robot T-Quad, développé par 3sigma. T-Quad est un AGV (Automated Guided Vehicle) qui possède  4 roues motrices omnidirectionnelles chacune commandées par son propre moteur électrique avec codeur incrémental (qui nous servira de capteur de position) et capteur de courant.




*Présentation du robot T-Quad*

Ces moteurs étaient commandés par une Raspberry Pi 3. Cependant, la notre était défectueuse: nous ne pouvions pas la connecter à internet, ce qui signifie que nous ne pouvions pas télécharger ROS, un système d'exploitation pour robots essentiel pour notre application. Nous nous sommes donc penchés vers une Raspberry pi 4.

Après avoir installé l'OS de la Raspberry et configuré internet, nous avons installé les packages ROS suivants :

| | |
|---|---|
| ros-kinetic-usb-cam | Fournit le flux vidéo de la caméra branché en usb. |
| ros-kinetic-rosbridge-server | Fournit une interface de connexion à client non Ros pour la communication |
| ros-kinetic-rosserial-arduino | Fournit un protocole de communication ROS pour l'arduino |

Nous avons téléchargé les bibliothèques et décompressé dans le répertoire des librairies de votre IDE arduino avant le téléversement du firmware arduino des T-Quads dans la carte Mega. suivantes

| Nom de la librairie | Utilité | Provenance |
|---|---|---|
| DigitalWriteFast | Ecrire plus rapidement sur les Entrées/Sorties de l'Arduino | http://www.3sigma.fr/telechargements/digitalWriteFast.zip |
| EnableInterrupt | Gère les interruptions sur l'Arduino | http://www.3sigma.fr/telechargements/EnableInterrupt.zip |
| NewPing | gère le capteur ultrason du robot T-Quad | http://www.3sigma.fr/telechargements/NewPing.zip |
| roslib | permet de faire le lien entre ROS et arduino | commande terminal: rosrun rosserial arduino make_librairies.py |

## C. Gestion des déchets

Pour ramasser les déchets, le robot est équipé d'un bac de récupération auquel est attaché un laser et une photorésistance. A l'aide d'un algorithme, nous sommes capable de savoir si un objet est passé entre le laser et la photorésistance et d'allumer une led correspondant à une situation respective :

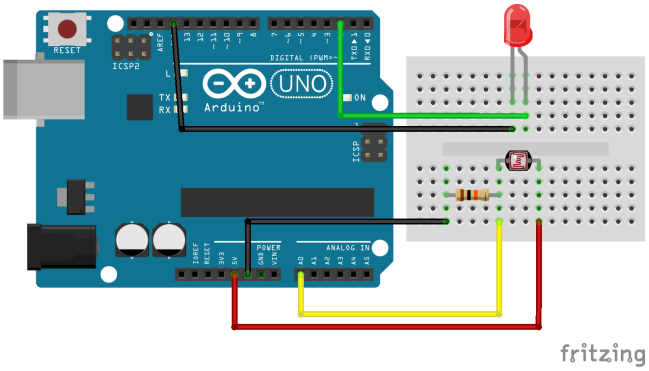



Schéma simplifié (en réalité nous avons 2 Leds et 1 laser)

### 1) Le bac est rempli

Nous savons si le bac est rempli lorsque le temps de coupure du faisceau laser sur la photorésistance est plus long que 1 seconde, cette action amène le robot à allumer une led 1 (bac plein)

### 2) Un objet est passé de façon brève entre le laser et la photorésistance

Nous savons si le robot à bien ramasser un déchet si le temps de coupure du faisceau laser est inférieur à 100ms, dans ce cas, le robot allume la led 2 (déchet bien ramassé)

Code de la gestion de déchets en arduino :

```
int lightPin = 0;
int ledPin1 = 9;
int ledPin2 = 10;
int bref =0;
int loong=0;

void setup(){
     pinMode (ledPin1, OUTPUT); //initialisation
     pinMode (ledPin2, OUTPUT);
     Serial.begin(9600);
}

void loop(){
     int seuil = 850;   //on initialise un seuil pour lequel la led
s'allumera
     bref=(analogRead (lightPin)< seuil) ;
     delay(20); //dans le cas d'une détection brève
     loong=(analogRead (lightPin)< seuil);
     if (bref==1 && loong==1){ //si la photorésistance //détecte
toujours le laser (plus de 20ms)         digitalWrite (ledPin1,
HIGH);  //on allume la led 1 pendant 1s
         delay(1000);
         }
     else if (bref==1 && loong==0) //si la photorésistance ne
//détecte plus le laser (soit moins de 20ms)
     {
         digitalWrite (ledPin2, HIGH); //on allume la led 2 pendant
1s
         delay(1000);
     }
     else{ //on éteint les leds ensuite
       digitalWrite (ledPin2, LOW);
       digitalWrite (ledPin1, LOW);
     }
 }
```

## D. Développement de l'IA

Durant ce semestre, nous nous sommes concentrés sur la compréhension et la recherche de comment créer une IA capable de détecter les mégots de cigarette et comment chaque partie va communiquer entre elles. Pour réaliser cette partie, notre tuteur de projet nous as prêté une **Nvidia Jetson Nano**, un micro-ordinateur de chez Nvidia très utilisé dans le développement d'IA :

Pour développer l'IA, nous nous inspirons des vidéos expliquant le fonctionnement et la façon de faire présent sur la chaîne youtube ***Nvidia Developper*** qui est la chaîne youtube officielle des développeurs Nvidia ainsi que le dépôt Git associé :

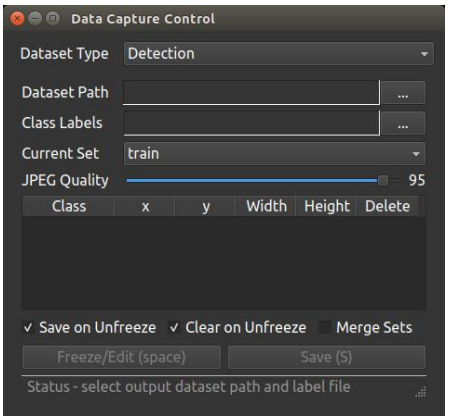
*https://github.com/dusty-nv/jetson-inference*

Pour développer une IA qui reconnaît des mégots de cigarette, nous devons tout d'abord l'entraîner. Pour comparer, notre IA fonctionne de la même façon que n'importe quel apprentissage : en répétant les entraînements pour mieux apprendre.

Pour faire ces entraînements, nous devons donc créer une base d'image de mégots de cigarette répertorié en 3 dossiers :

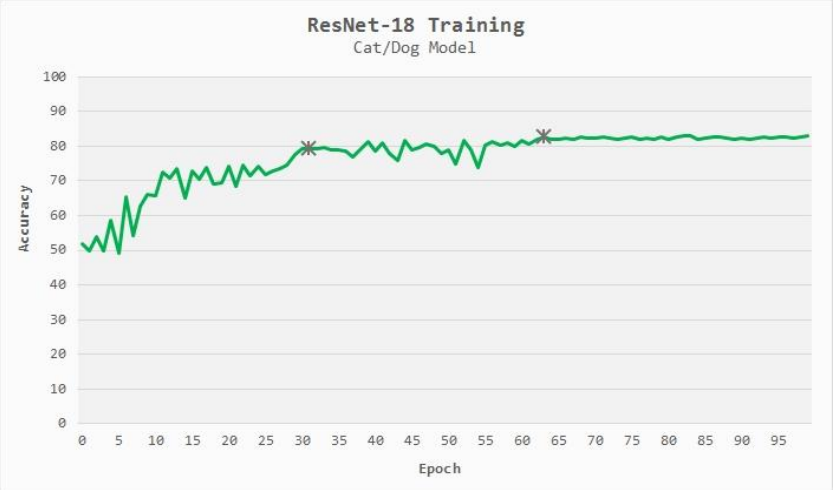
- **/train** : Ce sont les données qui vont entraîner notre modèle
- **/test** : Ce sont les données qui vont tester notre modèle
- **/val** : Ce sont les données qui vont valider notre modèle

Pour faciliter le travail, le git associé dispose d'un logiciel permettant de créer automatiquement les dossiers et d'y ranger les images prises directement via une Webcam :




Lorsque nous allons procéder à l'entraînement de notre modèle, l'IA va passer par chaque image de chaque dossier en commençant par le dossier **/train** pour s'entraîner, puis par le dossier **/test** pour se tester et enfin par le dossier **/val** pour se valider. Un passage dans l'ensemble des données est appelé **epoch**. A la fin d'un epoch, nous obtenons donc une IA capable de reconnaître un mégot de cigarette avec une certaine précision. Le nombre d'epoch ainsi que le nombre d'images a une incidence sur la précision obtenue.

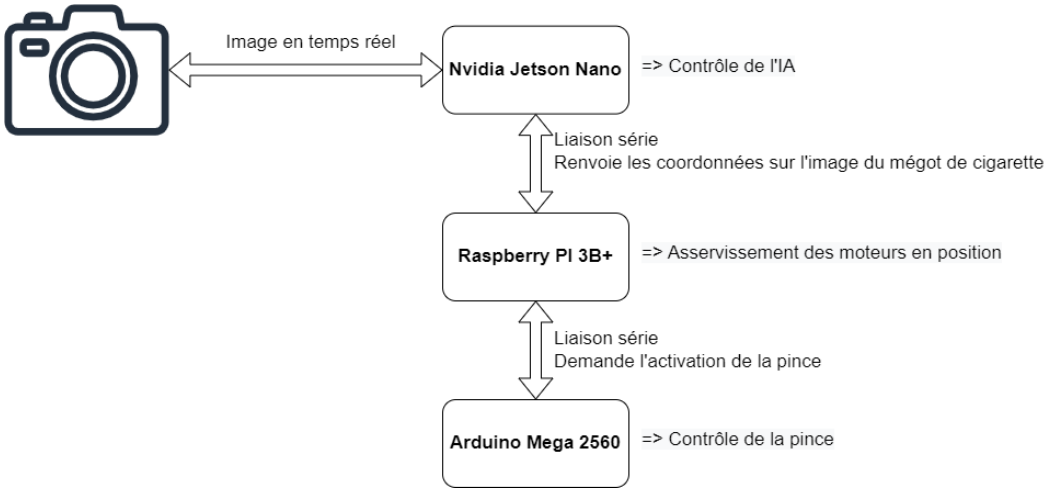
Le Git associé nous donne un exemple avec une base d'image de 5000 images de chien et de chat :




Nous remarquons donc que pour 5000 images, nous tendons vers une précision de 80% à partir de 30 epochs.

Après cette partie d'analyse, nous avons récolté des mégots de cigarette que nous allons utiliser pour la création de la base d'image puis nous allons créer notre modèle et le tester.

En parallèle de cette analyse, nous avons aussi réfléchi sur comment les différentes parties ainsi que les différents modules allaient communiquer entre eux :




Il est totalement possible de regrouper les fonctions de la Nvidia Jetson Nano et de la Raspberry sur le même module pour du gain de place mais nous restons sur cette configuration pour pouvoir travailler sur les différentes fonctions en même temps.

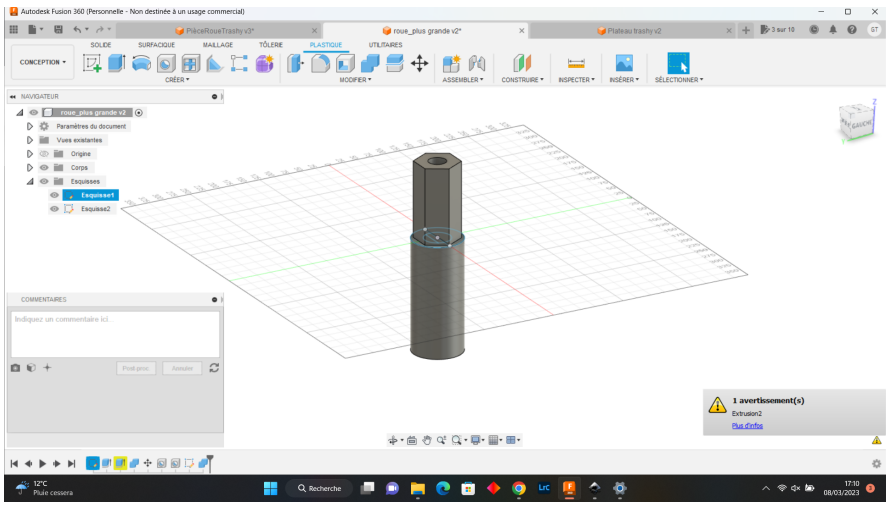
# III. Semestre 8

Lors de ce semestre nous avons finalisé notre preuve de concept. Cependant, nous avons rencontré beaucoup de soucis techniques. En effet, l'un des composants de notre base mobile a grillé suite à une erreur de notre part accumulée à une fatigue de la base mobile. Cela a impliqué de devoir complètement remplacer notre base mobile et notre carte de contrôle des moteurs.

# A. Mobilité du robot

Ayant juste un chassis en métal prétroué, des moteurs et des roues mécanums, nous avons dû apporter quelques modifications afin de pouvoir connecter les roues aux moteurs (eux mêmes reliés au châssis)

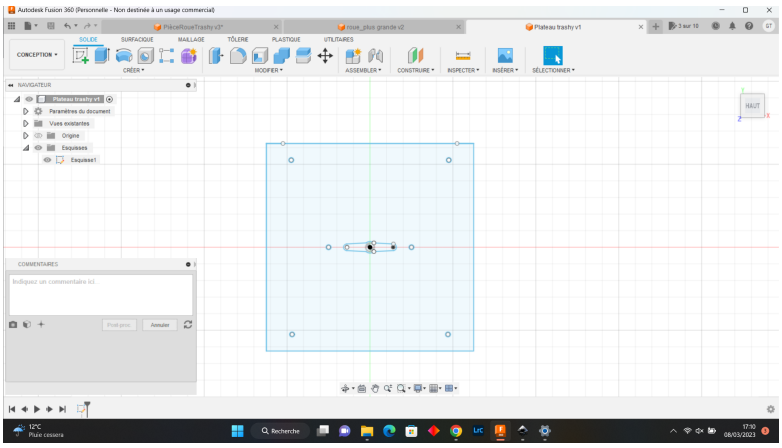
Les roues que nous avons à notre disposition ne s'adaptent pas aux moteurs qui nous avaient été fournis. Nous avons donc dû créer une pièce en 3D afin de convertir la forme hexagonale en forme de cercle coupé qui s'adapte au moteur



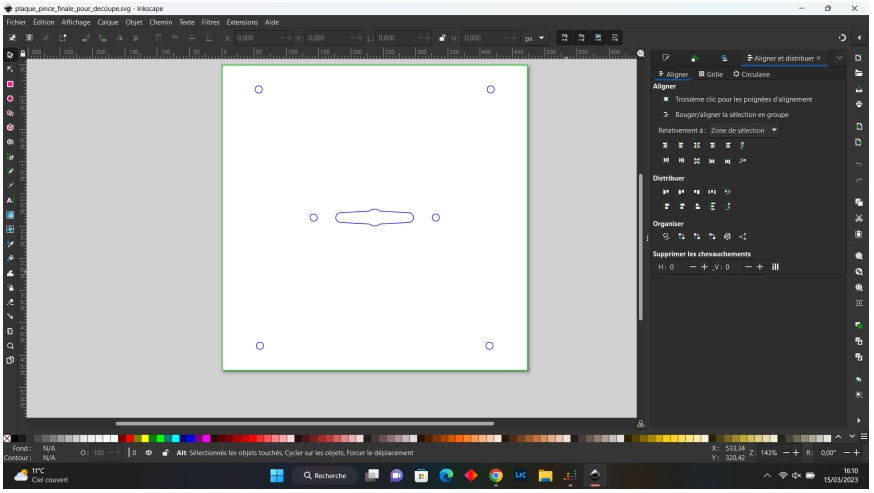
Pièces en 3d créé dans le logiciel fusion 360

Pièce en 3d une fois connecté à la roue

De plus, le support de la pince, que nous avions à notre disposition depuis le premier semestre, était beaucoup trop grand comparativement à la taille de base du châssis, nous avons donc décidé de recréer une base qui s’adapte au châssis.

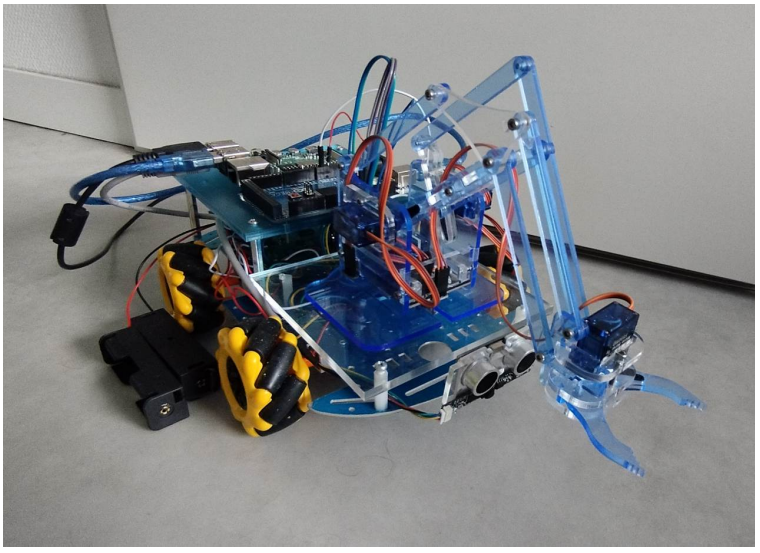


Esquisse de la base dans le logiciel fusion 360



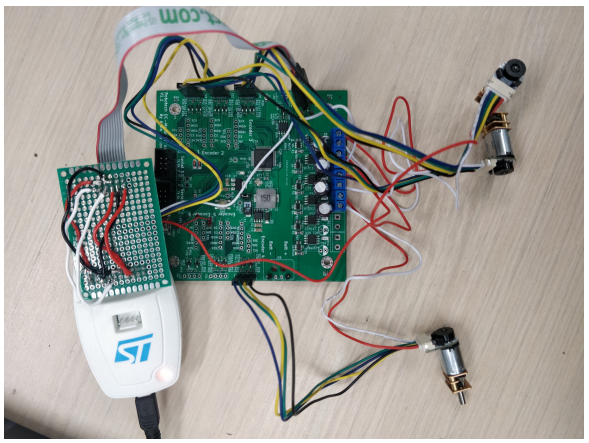
Base prête à la découpe dans le logiciel inkscape

Afin de palier au problème rencontré avec notre base mobile, nous avons dû en recommander une et utiliser la carte de contrôle moteur confectionnée par Albin MOUTON du projet “Coupe de France de Robotique” de notre promotion :



*Nouvelle base mobile + carte de contrôle moteur*

Nous avons du téléverser le code du  driver moteur dans la carte (aimablement fourni par Albin MOUTON également).



La carte de contrôle moteur permet d'assigner une vitesse de rotation à chaque moteur et de la garder juste grâce à un correcteur PID qui utilise l'encodeur des moteurs. Pour la communication vers la carte, elle fonctionne grâce à une liaison série avec la RaspBerry et l'utilisation des G-Codes (aussi utilisé dans les imprimantes 3D) :

| Commande | G[n] | Paramètres | Retour |
|---|---|---|---|
| Définir la commande de Vitesse | G0 | [X… Y… Z… U… V… W…] : vitesse en tr/min | / |
| Définir la commande de Courant | G1 | [X… Y… Z… U… V… W…] : courant en A | / |
| Définir la commande de PWM | G2 | [X… Y… Z… U… V… W…] : PWM de sortie (de -4095 à 4095) | / |
| Lire la mesure de vitesse | G10 | [X Y Z U V W] : demande de la dernière lecture de la vitesse d'un axe (tr/min) | G10 [X… Y… …] |
| Lire la mesure de courant | G11 | [X Y Z U V W] : demande de la dernière lecture de courant d'un axe (A) | G10 [X… Y… …] |
| Définir le gain du PID de vitesse | G20 | P: Kp<br>I : Kp<br>D: Kd<br>M:Moteur concerné par le changement des gains | / |
| Définir le gain du PID de courant | G21 | P: Kp<br>I : Kp<br>D: Kd<br>M:Moteur concerné par le changement des gains | / |
| Définir le mode de régulation | G22 | R: Type de régulation (1=ouverte, 2=courant, 3=vitesse, 4=position)<br>M:Moteur concerné par le changement des gains | / |

Nous pouvons donc définir la vitesse des moteurs avec la commande suivante :

```
G0 X40 Y40 Z40 U40
```

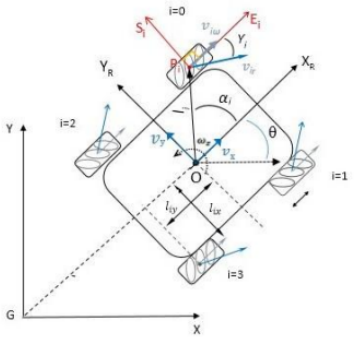
Et lire la valeur de la vitesse des moteurs avec la commande :

```
G10 X Y Z U
> G10 X48.529 Y35.588 Z38.824 U42.059
```

Maintenant que nous avons un moyen de contrôler les moteurs séparément, nous devons développer un programme permettant la mobilité du robot. Pour ce faire, nous nous basons du document[1] faisant étude de la cinématique et des formules rentrant en jeu pour le mouvement d'un robot à 4 roues omnidirectionnelles et des formules suivantes :

**2. KINEMATIC**

Figure 2 shows the configuration of a robot with four omnidirectional wheels.




$$\begin{bmatrix}\omega_1\\ \omega_2\\ \omega_3\\ \omega_4\end{bmatrix} = \frac{1}{r}\begin{bmatrix}1 & -1 & -(l_x+l_y)\\ 1 & 1 & (l_x+l_y)\\ 1 & 1 & -(l_x+l_y)\\ 1 & -1 & (l_x+l_y)\end{bmatrix}\begin{bmatrix}v_x\\ v_y\\ \omega_z\end{bmatrix}. \qquad \text{eq. 19}$$

$$\begin{cases}\omega_1 = \frac{1}{r}(v_x - v_y - (l_x+l_y)\omega),\\ \omega_2 = \frac{1}{r}(v_x + v_y + (l_x+l_y)\omega),\\ \omega_3 = \frac{1}{r}(v_x + v_y - (l_x+l_y)\omega),\\ \omega_4 = \frac{1}{r}(v_x - v_y + (l_x+l_y)\omega).\end{cases} \qquad \text{eq. 20}$$

Dans la formule 20, la variable r correspond au diamètre de notre roue et Lx et Ly correspondent aux largeurs et longueurs du robot. Nous programmons ainsi la fonction suivante :

```python
def translation(Vx, Vy, Wz):
  """
      Define the parameter for the G-Code to make a movement
      depending on the speed Vx, Vy, Wz
      :param Vx: (float) Speed on the X axis
      :param Vy: (float) Speed on the Y axis
      :param Wz: (float) Speed on the Z axis
      :return: A float-list of size 4 with each parameter for the G-Code
[X, Y, Z, U]
      CU : None
  """
  X = (1/R) * (Vx - Vy - (Lx + Ly)*Wz)
  Y = (1/R) * (Vx + Vy + (Lx + Ly)*Wz)
  Z = (1/R) * (Vx - Vy + (Lx + Ly)*Wz)
  U = (1/R) * (Vx + Vy - (Lx + Ly)*Wz)
  print("Entrée : < Vx =",Vx,"| Vy =",Vy,"| Wz =",Wz," >")
  print("Sortie : < X =",X,"| Y =",Y,"| Z =",Z,"| U =",U," >")
  return [X, Y, Z, U]
```

*Fonction permettant la conversion de la vitesse du robot vers la vitesse des moteurs*

[1] https://research.ijcaonline.org/volume113/number3/pxc3901586.pdf

Maintenant que nous arrivons à déterminer la vitesse à donner à chaque moteur en fonction du déplacement que l'on veut faire, nous devons les envoyer sous le bon format à la carte de contrôle moteur en utilisant la librairie "Serial" de python. Nous devons d'abord initialiser la connection :

```
# Initialization of Motor_controller serial connection

port_Moteur = "/dev/ttyACM0"
baudrate_Moteur = 115200

moteur = serial.Serial(port_Moteur,baudrate_Moteur,timeout=1)
```

*Initialisation de la liaison série*

Nous écrivons ensuite une fonction permettant d'envoyer sur la liaison série, le G-Code correspondant :

```
def send_G_Code(w):
  command = "G0 X{:.3f} Y{:.3f} Z{:.3f} U{:.3f}\n".format(w[0],w[1],w[2],w[3])
  print("Command sent :",command)
  moteur.write(command.encode('ascii'))
```

*Fonction permettant l'envoie du G-Code*

## B. Reconnaissance des objets

Pour la partie de la reconnaissance des mégots de cigarette, nous étions partis à la base sur le développement d'une IA de reconnaissance d'image en utilisant la librairie TensorFlow.

Cependant, après de nombreux essais et développement, nous n'avons pas réussi à produire une IA permettant la reconnaissance des mégots de cigarette. Afin de pouvoir travailler la partie mobilité et le programme permettant à Trashy de se diriger vers le mégot de cigarette, nous avons développé un moyen de reconnaître les objets oranges sans pour autant pouvoir les classifier.

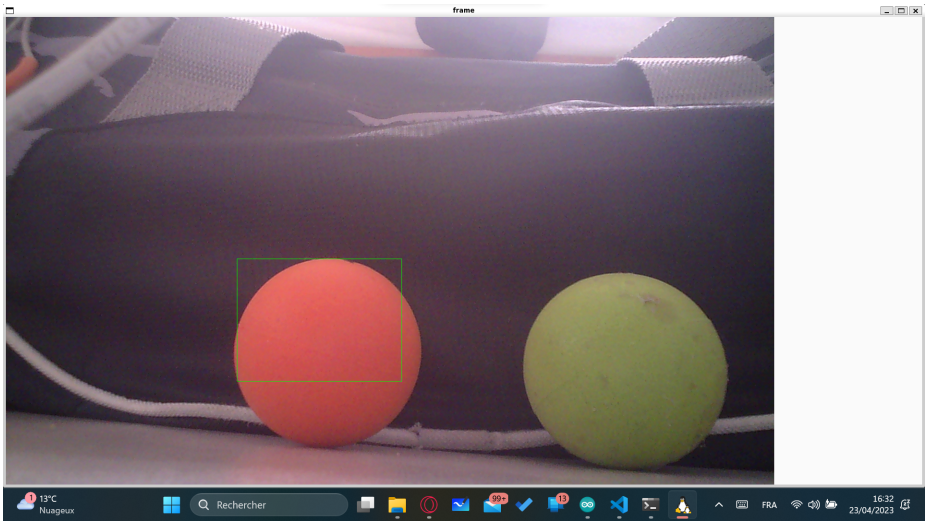
Pour ce faire, nous utilisons la librairie OpenCV de Python qui permet d'analyser les images perçues par la caméra. Nous écrivons donc une fonction permettant de reconnaître les objets oranges et de dessiner un rectangle autour de cet objets en récupérant les coordonnées x,y du coin supérieur gauche ainsi que sa largeur et hauteur :

```
    cap = cv2.VideoCapture(0)
    while(cap.isOpened()):
        ret, frame = cap.read()
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        light_orange = (0,90,200)
        dark_orange = (18,180,255)
```

```
            mask = cv2.inRange(hsv, light_orange, dark_orange)
            res = cv2.bitwise_and(frame,frame, mask= mask)
            contours, hierarchy = cv2.findContours(mask, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
            for cnt in contours:
                [x, y, w, h] = cv2.boundingRect(cnt)
                if w > 50:
                    print("< x :",x,"| y :",y,"| w :",w,"| h :",h,">");
                    cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 0)
```

*Programme permettant la reconnaissance d'objet orange*

On obtient le résultat suivant :




Le contour n'est pas parfait mais est suffisant pour le repérage. Pour notre preuve de concept, nous utiliserons donc ces petites balles de couleur afin de représenter les mégots de cigarette.

## C. Liaison entre les différents modules

Maintenant que nous avons toutes nos parties fonctionnelles, nous pouvons lier chaque partie entre elles.

### a. Liaison entre la RaspBerry et la pince

Afin de pouvoir commander la pince au bon moment, nous relions l'Arduino Mega 2560 qui commande la pince à la RaspBerry grâce à une liaison série. Nous modifions donc le code Arduino afin d'activer la pince à la réception du message 'o' via la liaison série :

```
void loop(){

  if(Serial.available() > 0) {
    incomingByte = Serial.read();

    Serial.print("I received :");
    Serial.println((char) incomingByte);
    if((char) incomingByte == 'o'){
      Serial.println("Je dois attraper");
      attrape_object();
    }
    delay(500);
  }
}
```

Nous testons donc la connexion entre la RaspBerry et l'arduino grâce à ce script Python qui utilise la librairie Serial afin de communiquer en série avec l'arduino :

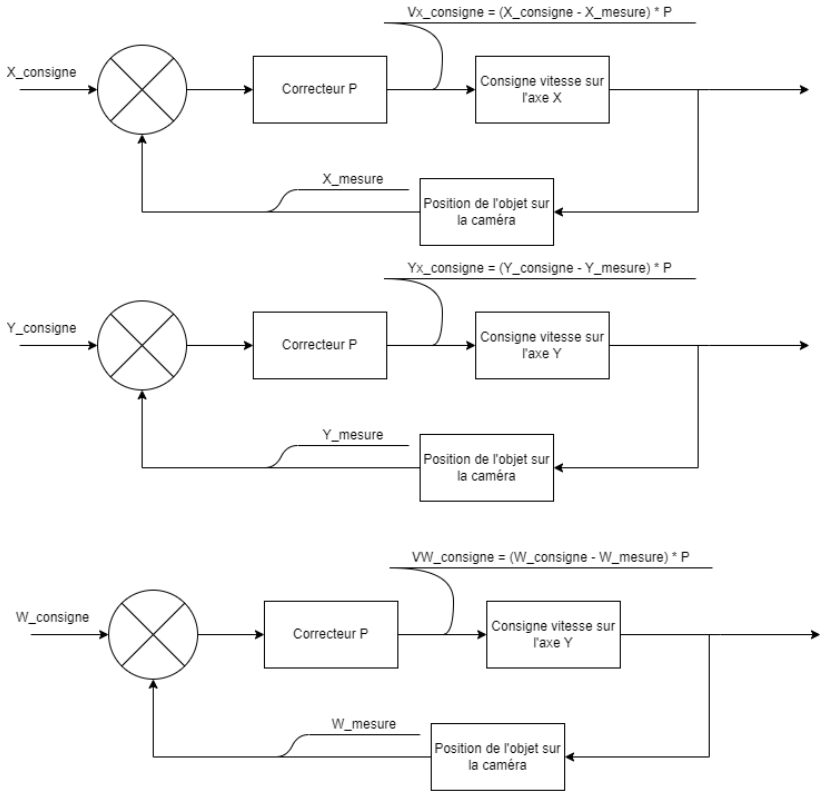
```
import serial
import time

port = "/dev/ttyACM0"
baudrate = 9600

arduino = serial.Serial(port,baudrate)

print("Enter 1 to test :")
x = input()
print(x)

if x=='1':
  print("On envoie")
  arduino.write(b'o')
  time.sleep(2)
  x = 7
```

Grâce à ce script, nous pouvons donc tester la bonne connexion entre la RaspBerry Pi 4 et l'Arduino servant au contrôle de la pince.

## b. Liaison entre la reconnaissance d'objet et la mobilité

Afin de lier la reconnaissance d'objet et la mobilité du robot, nous allons asservir la position de la balle orange sur la caméra pour qu'elle soit à l'endroit propice afin que la pince puisse l'attraper.

Nous schématisons ainsi la boucle d'automatisation suivante :




Nous allons donc utiliser une régulation P afin de mieux contrôler la vitesse injectée aux moteurs. Après différents tests, nous obtenons P = 0.15.

Afin de contrôler notre robot, nous regardons d'abord si les paramètres x,y et w du rectangle entourant la balle orange sont dans les bornes propices au ramassage de la balle par la pince. Une fois que le robot est bien placé devant la balle, nous pouvons la ramassé :

```
if not((x > X_min) and (x < X_max)):
      Vx_consigne = (X_consigne - x) * P
      print("Vx_consigne :",Vx_consigne)
      action = subprocess.run(["python3",Motor,"0",str(Vx_consigne),"0"])
elif not((y > Y_min) and (y < Y_max)):
      Vy_consigne = (y - Y_consigne) * P
      print("Vy_consigne :",Vy_consigne)
      action = subprocess.run(["python3",Motor,str(Vy_consigne),"0","0",])
elif not((w > W_min) and (w < W_max)):
      Vw_consigne = (W_consigne - w) * P
      print("Vw_consigne :",Vw_consigne)
```

```
        action = subprocess.run(["python3",Motor,str(Vw_consigne),"0","0",])
else:
        print("Catch sequence")
        action = subprocess.run(["python3",Motor,"0","0","0"])
        time.sleep(2)
        to_catch = True
        break
```

Maintenant que nous avons la méthode de fonctionnement, nous allons régler les paramètres des bornes afin que notre robot s'arrête devant la balle et puisse l'attraper. Après des tests, nous obtenons les réglages suivants :

```
# Variable
X_consigne = 300
X_min = 220
X_max = 380

Y_consigne = 370
Y_min = 270
Y_max = 470

W_consigne = 670
W_min = 570
W_max = 770

P = 0.15
```

# IV. Pour aller plus loin

Notre robot est pour l'instant capable de se diriger vers une balle orange assez proche d'elle et de l'attraper mais toute la partie de reconnaissance d'image et de mégot de cigarette n'est malheureusement pas fonctionnelle. Une piste d'amélioration serait donc de développer une IA fonctionnelle.

Il reste également quelques fonctionnalités à développer pour que le projet soit commercialisable, notamment la géolocalisation du robot ainsi que la capacité à se déplacer vers une zone de dépôt une fois le réservoir de mégots plein.

# Conclusion

Nous avons presque pu créer un robot ramasseur de mégots ayant un niveau de maturité équivalent à un milieu de TRL4 (Test of Readiness Level).

De plus, ce projet nous a permis de développer des compétences humaines et sociales notamment de répartitions des tâches et d'entraide, de modélisation 3D et des compétences de création d'Intelligence Artificielle. Nous avons également appris à gérer une crise au cours d'un projet ainsi que les imprévus lorsque certaines étapes du projet initial n'est pas fonctionnelle.

# Remerciement

Nous aimerions remercier notre tuteur de projet, Mr Othman Lakhal, ainsi que les professeurs de Polytech dont Mr Thierry Flamen pour son aide sur le côté électronique lorsque nous avons rencontré notre problème sur notre première base mobile. Nous aimerions aussi remercier Albin Mouton en SE4 pour sa carte de contrôle moteur qui nous a bien servi de remplacement.