Rapport de l’équipe Wall-E 2.0

Dans ce rapport, nous mettrons en avant les difficultés que nous avons pu rencontrer durant la construction et la programmation de notre robot, et les solutions que nous avons trouvées.

I. Construction mécanique

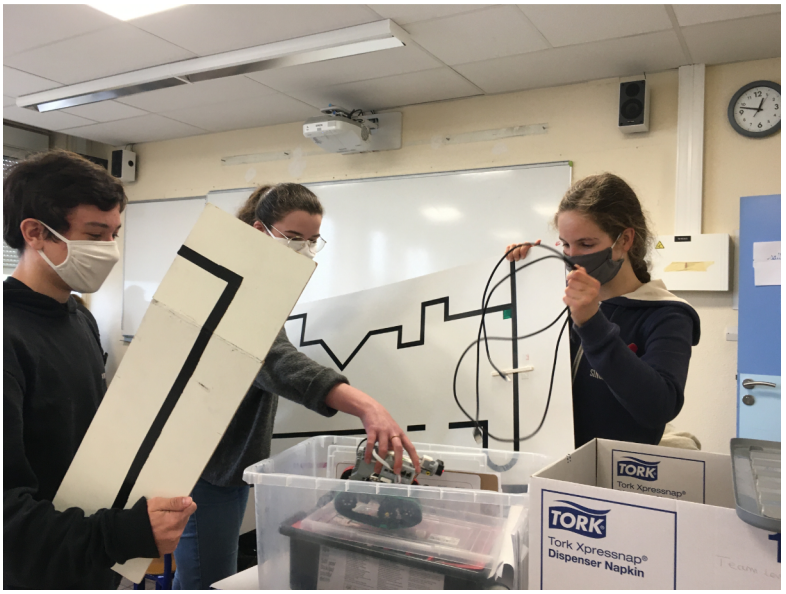
Tout d’abord, il y a eu un problème d'**adhérence des chenilles** : nous avons utilisé des chenilles composées de pièces en plastique rigide qui glissaient. Nous avons d’abord tressé dans les chenilles des fils en caoutchouc découpés dans des tuyaux de chimie, mais ils faisaient dérailler les roues. Nous avons finalement doté notre robot de gros élastiques en caoutchouc. Ces élastiques étaient trop serrés et empêchaient la roues de tourner correctement. Donc, nous les avons placés sur une planche pour qu’ils se détendent.

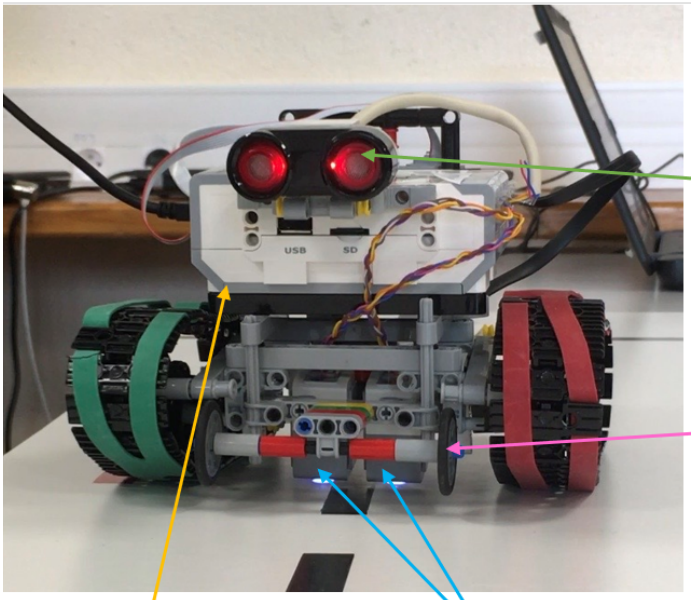
La **répartition du poids** a aussi été problématique notamment pour les obstacles tels que le pont basculant ou la pente, le robot avait tendance à basculer lors de la descente. Notre solution fut de rajouter une roue (mobile) sur le devant pour l'empêcher de basculer.

Nous avons rencontré un problème d’**éloignement des capteurs de couleurs avec le sol** : ces capteurs étaient trop loin du sol pour bien détecter toutes les couleurs, mais après, trop près pour pouvoir rouler sur des petits obstacles (barre de lego). Nous avons donc mis au point un système ajustable pour pouvoir éloigner, de la longueur désirée seulement, les capteurs.

Nous avons aussi fabriqué nos propres **câbles** pour relier la brique à tous les capteurs et moteurs.

Finalement, nous nous sommes amusés à personnaliser notre robot, notamment en choisissant les couleurs des élastiques permettant aux chenilles d’adhérer au sol. Nous avons mis du rouge à gauche et du vert à droite, si on le regarde de derrière, ce qui correspond aux mêmes couleurs que pour les bateaux ou les avions.

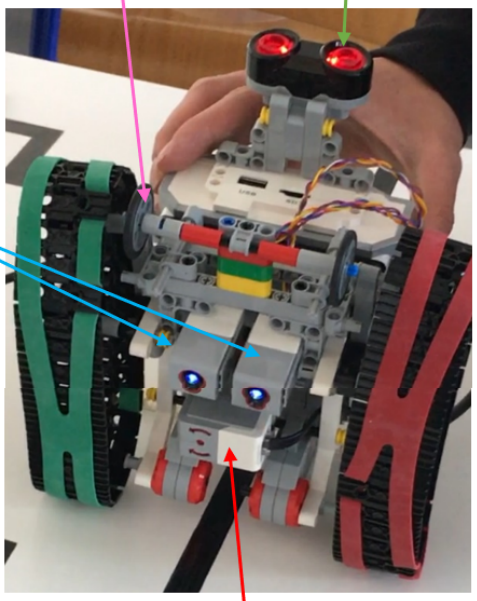




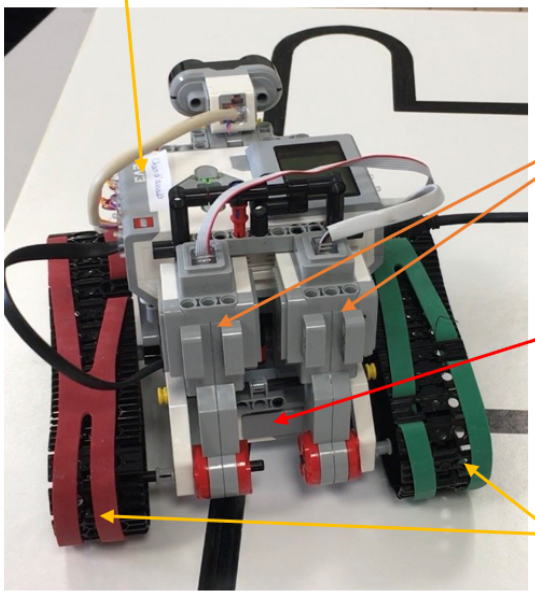
Capteur à ultrasons

Roue pour ne pas basculer

Capteurs de couleur



Brique EV3



Moteurs

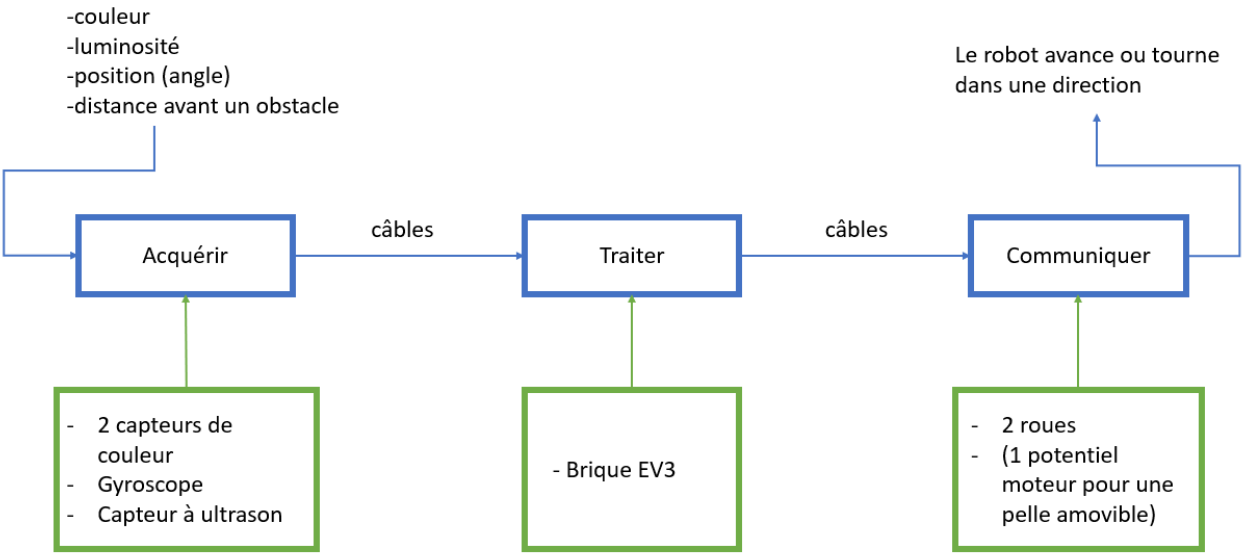
Gyroscope

Chenilles (rouge à bâbord, vert à tribord)

II. Programmation

1. Chaîne d'information et base du programme

Voici la chaîne d'information de notre robot :



Nous avons codé la base du programme à partir du modèle d'un des professeurs auquel nous avons apporté notre touche pour l'améliorer à l'aide de l'expérience acquise grâce à notre participation à la Robocup Junior depuis la classe de Seconde. Puis nous avons ensuite ajouté toutes les autres fonctions pour lui permettre de gérer plus de situations.

Lors des essais pour affiner le programme et tester s'il marchait correctement, nous avons fait fonctionner le robot accroché à l'ordinateur via un câble usb pour réduire le temps qu'il prenait à charger le programme (c'est pour cela que l'on voit un câble pendant la vidéo). Mais le robot peut aussi très bien avancer tout seul une fois le programme chargé.

2. Définition des couleurs

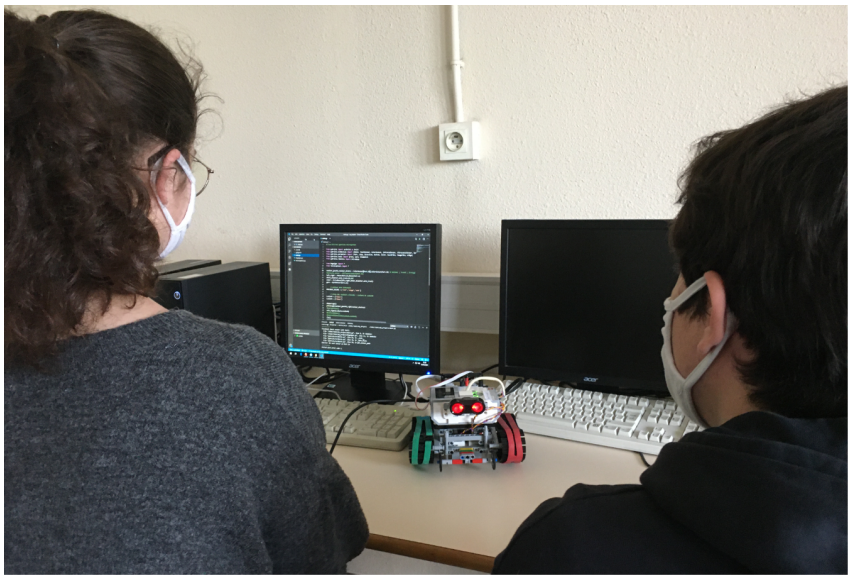
Nous avons eu du mal à définir les couleurs précisément. En effet, parfois, à la place d'identifier du blanc comme du blanc, notre programme l'indentifiait parfois comme du vert, ce qui déclenchait une action non désirée. Pour résoudre ce problème, nous avons mélangé les données sur la luminosité aux données RGB (qui correspondent aux intensités des trois couleurs : rouge, vert et bleu), toutes deux obtenues via les capteurs de couleurs, il fallait donc que la luminosité et le RGB soient tous les deux conformes pour identifier une couleur, ce qui limitait la marge d'erreur. Nous avons aussi réduit le champ des valeurs de luminosité pour lesquelles la couleur était identifiée comme du vert.

A partir de la définition de ces couleurs, grâce à une fonction que nous avons appelée get-color prenant en argument un capteur et effectuant des tests sur la couleur captée avec

son code RGB et sa luminosité pour renvoyer sous la forme d'un string la couleur captée, nous avons pu associer une ou plusieurs commandes à une couleur ou enchainement de couleurs. La mémorisation des couleurs permet au robot d'effectuer la bonne action lorsqu'il rencontre du vert. S'il y avait du blanc avant, il tourne, alors que si il y avait du noir, il ne prend pas en compte le vert et continue son action.

3. Structure du programme

Le programme tourne en boucle dans la fonction principale "suit_ligne" qui permet, comme son nom l'indique, de suivre la ligne en testant et en mémorisant continuellement les couleurs à droite et à gauche, et en vérifiant grâce au capteur à ultrason qu'il n'y a pas d'obstacle devant, pour effectuer les actions nécessaires pour rester sur la ligne et avancer. Parmi ces actions, nous avons paramétré des fonctions qui permettent de réaliser les tâches les plus longues et répétitives. Il y a par exemple la fonction "tourne" qui prend un angle en argument et tourne jusqu'à atteindre cet angle, ou la fonction "obstacle_contournement" (qui effectue les mouvements nécessaires pour contourner un obstacle)...



Voici un extrait du code (ce n'est pas la dernière version) :

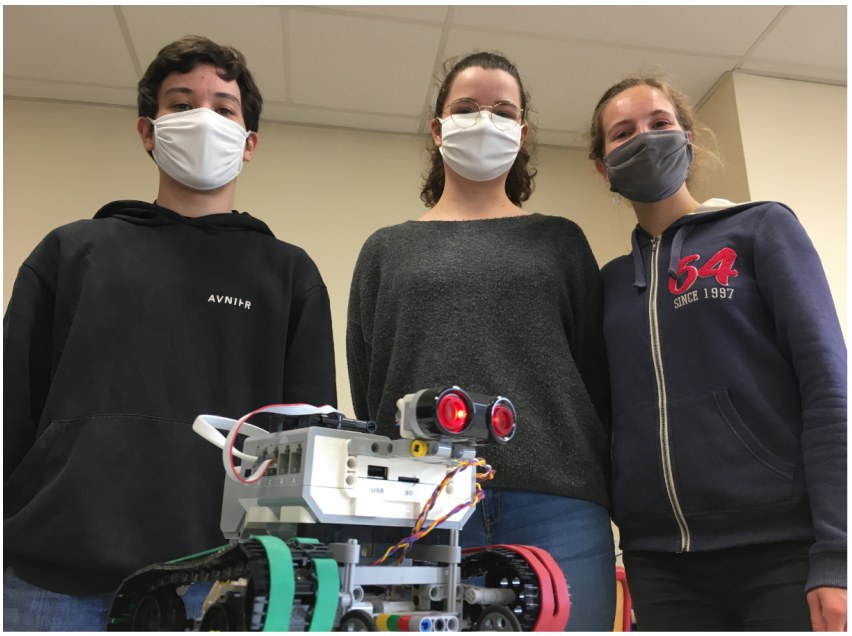
```
60
61  def tourne(limite) :
62      gyro.reset_angle(0)
63      if (limite == limite_demitour) : pass #brick.sound.file(SoudFile.FORWARD)
64      else :
65          brick.sound.file(SoudFile.GREEN)
66          robot.drive_time(speed, 0, 600)
67      if limite < 0 :
68          while gyro.angle > limite : robot.drive_time(0, -45, 200)
69      else :
70          while gyro.angle < limite : robot.drive_time(0, 45, 200)
71      print(gyro.angle())
72
73  def suit_ligne() :
74      print("suitlinge")
75      lg = [None, None, get_color(color_left)]
76      ld = [None, None, get_color(color_right)]
77      print(lg,ld)
78      while lg[-1] != "rouge" or ld[-1] != "rouge" :
79          robot.drive_time(speed, 0, time)
80          color_g, color_d = get_color(color_left), get_color(color_right)
81          if color_g != lg[-1] : lg.append(color_g)
82          if color_d != ld[-1] : ld.append(color_d)
83
84          if color_g == "blanc" and color_d == "noir" :
85              if ld[-2] == "vert" and ld[-3] == "blanc" :
86                  tourne(limite_droite)
87              else :
88                  robot.drive_time(0, -45, 200)
89                  robot.drive_time(-speed, 0, 50)
90          elif color_g == "noir" and color_d == "blanc" :
91              if lg[-2] == "vert" and lg[-3] == "blanc" :
92                  tourne(limite_gauche)
93              else :
94                  robot.drive_time(0, 45, 200)
95                  robot.drive_time(-speed, 0, 50)
96          elif color_g == "noir" and color_d == "noir" : # y'avait marquait "à compl
```

On utilise cette fonction pour tourner jusqu'à l'angle limite

Le robot ne s'arrête pas de tourner tant que l'angle effectué est inférieur à celui voulu

Le programme met à jour les couleurs obtenues

Le robot effectue différentes actions selon les couleurs vues



Axelle Rolland, Samuel Vedel, Chloé Vagner

Merci à Mme Rouvière, Mme Gravelines et M Dulaurans