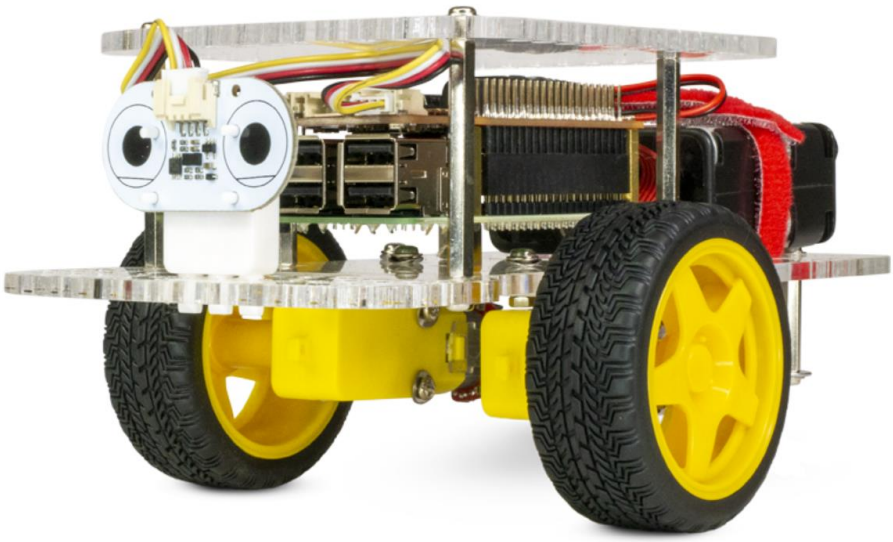
GP221 – Grand Projet de programmation

Sujet : **Stationnement autonome des automobiles**



Projet encadré par M. SADOUN Rabah

Table des matières

Résumé de la problématique 3

Contextualisation 3

Etat de l'art 3

Solutions possibles 4

Modélisation de la solution 5

Implémentation/validation 7

Conclusion 8

Résumé de la problématique

L'objectif principal de ce projet est de développer un code permettant à notre véhicule de stationner de manière autonome dans la configuration de stationnement en bataille. Cependant, nous sommes confrontés à un défi technique majeur, car notre robot, le GoPiGo3, ne dispose que d'un seul capteur lidar.

Contextualisation

Selon Ornikar, une étude révèle que 88% des automobilistes français accordent une grande importance à leur technique de stationnement. Malheureusement, les manœuvres de stationnement peuvent parfois s'avérer complexes et potentiellement accidentogènes, ce qui se traduit souvent par un mauvais positionnement de notre véhicule. Cependant, avec l'avènement de la robotique, l'automatisation de cette tâche devient envisageable. Qu'il s'agisse de stationnement en créneau, en bataille ou en épi, ces aspects sont désormais pris en considération par les constructeurs automobiles. La quête de la voiture totalement autonome est un impératif, allant de la prise en charge initiale du trajet jusqu'à la phase de stationnement.

La voiture autonome offre de nombreux avantages, notamment l'augmentation de la mobilité pour toutes les catégories de la population, y compris les personnes handicapées. Elle simplifie également la conduite lors des déplacements et des manœuvres de stationnement, contribuant ainsi de manière significative à la réduction du nombre d'accidents routiers et de leur gravité. En outre, elle améliore le confort des passagers lors de longs voyages.

Bien que les voitures totalement autonomes ne soient pas encore pleinement acceptées dans la plupart des pays, des constructeurs tels que Mercedes et Tesla se positionnent en tant que leaders de l'innovation dans ce domaine, en proposant des performances toujours plus fiables en matière de sécurité routière...

Etat de l'art

Webographie :

https://www.ornikar.com/code/cours/mecanique-vehicule/technologie-assistance/park-assist
https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9183957
https://patentimages.storage.googleapis.com/96/b7/d2/65e3f6afcc7b88/US20170329346A1.pdf
https://www.johndcook.com/blog/2013/05/05/ramanujan-circumference-ellipse/
https://www.turbo.fr/actualite-automobile/video-le-systeme-de-conduite-autonome-de-tesla-fait-tres-peur-184026#:~:text=En%20option%2C%20Tesla%20propose%20son,capacit%C3%A9%20de%20conduite%20enti%C3%A8rement%20autonome
https://fr.wikipedia.org/wiki/Rayon_de_courbure

Solutions possibles

Pour réaliser un stationnement en bataille avant, notre véhicule doit effectuer une manœuvre en forme d'ellipse. Pour ce faire, il est nécessaire de créer un modèle de trajectoire elliptique spécialement adapté à notre GoPiGo. En fonction des dimensions de notre robot, nous devons identifier un espace de stationnement ayant une largeur minimale de 16 cm. Dans le but d'optimiser la position du servo, nous envisageons de le placer au centre de cet espace, soit à une distance de 8 cm par rapport au point de référence O.

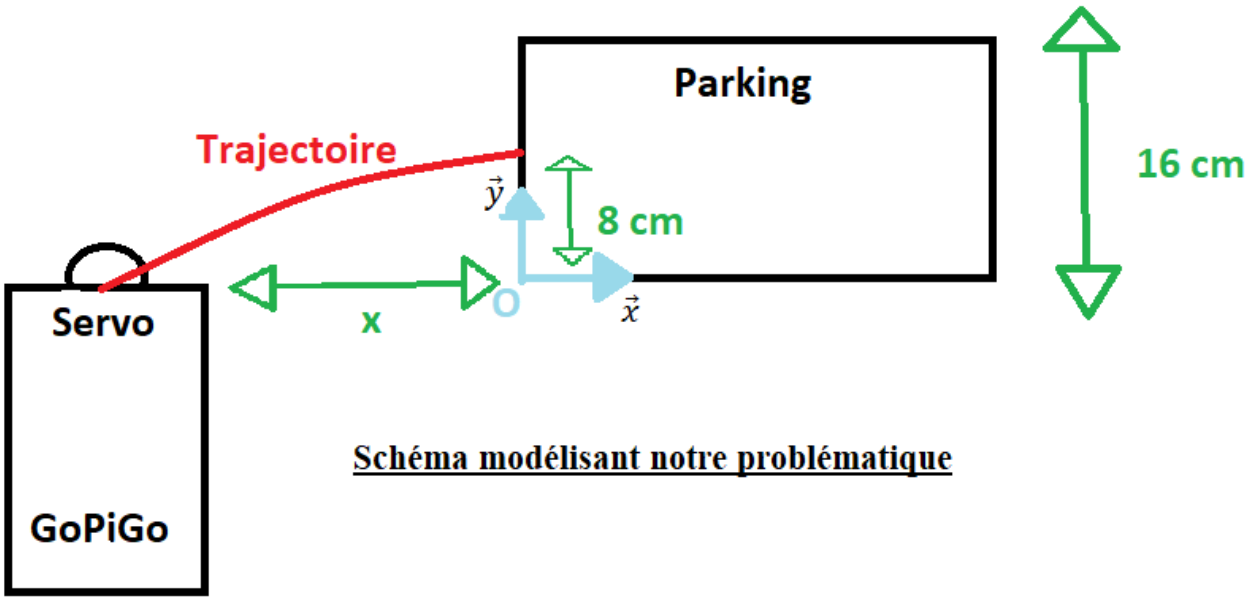


Schéma modélisant notre problématique

Pour répondre à ce défi, nous avons exploré plusieurs solutions possibles :

1. Modifier la trajectoire du robot en ajustant les rotations des roues.

Nous avons envisagé de moduler le taux de rotation de chaque roue afin de guider le robot le long de la trajectoire souhaitée. Cependant, nous avons été confrontés à une difficulté majeure : nous n'avions pas de méthode claire pour traduire une fonction en fonction de x permettant de réguler les rotations individuelles des roues afin de suivre précisément la trajectoire souhaitée. De plus si l'on confronte cette solution à un cas réel, il n'est pas encore évident de contrôler la rotation de chaque roue de manière indépendante.

2. Tracer la trajectoire elliptique à partir des rayons de courbure.

Étant donné qu'aucune fonction native du module easygopigo3 ne permet de déplacer le robot le long d'une trajectoire elliptique, nous avons développé une approche consistant à approximer une ellipse en utilisant des rayons de courbure. Plus précisément, nous avons décidé d'utiliser la fonction gopigo.orbit(angle, rayon) pour exécuter une séquence d'arcs de cercle successifs. Cette stratégie nous permet de créer une trajectoire qui se rapproche au mieux de l'idéal elliptique, ce qui permettra au robot d'effectuer un stationnement autonome en suivant cette trajectoire.

Modélisation de la solution

Pour concrétiser notre solution nous avons d'abord chercher la formule mathématique du rayon de courbure pour un arc dans le plan.

On donne d'abord l'équation d'une ellipse :

$$y(x) = \sqrt{\left(1 - \left(\frac{x}{a}\right)^2\right) * b^2}$$

Avec a le demi grand axe de l'ellipse et b le demi petit axe.

On la dérive une première fois :

$$y'(x) = \frac{-x * b^2}{y(x) * a^2}$$

Puis une seconde fois :

$$y''(x) = \frac{\frac{x^2 * b^4}{y(x)} + a^2 * b^2 * y(x)}{a^2 * y(x)^2}$$

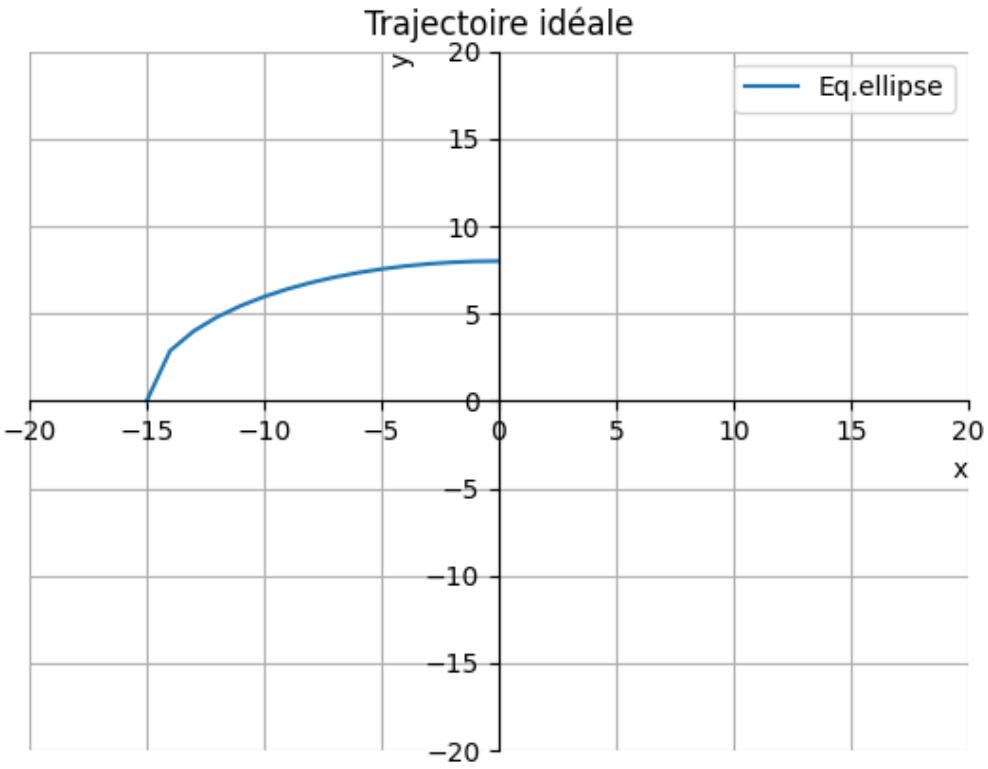
D'après nos recherches, nous trouvons l'équation suivante pour le rayon de courbure en x :

$$r(x) = \left|\frac{(1 + y'(x)^2)^{\frac{3}{2}}}{y''(x)}\right|$$

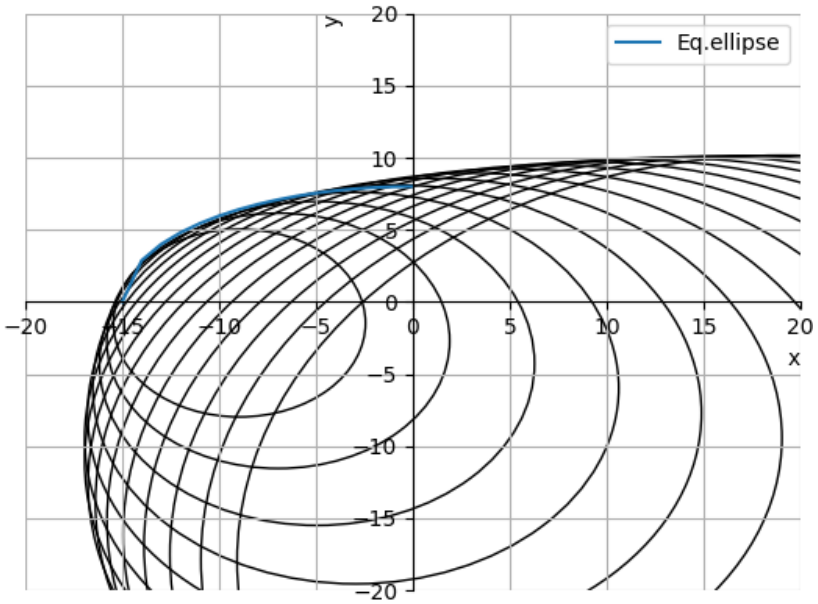
En python cela donne :

```
def fct_yx(self, x, a, b):
    return sqrt((1 - (x/a)**2) * b**2)

def fct_der1_yx(self, x, a, b):
    return (-x * b**2) / (a**2 * self.fct_yx(x, a, b))

def fct_der2_yz(self, x, a, b):
    return ((x**2 * b**4 / self.fct_yx(x, a, b)) + (b*a)**2 * self.fct_yx(x, a, b)) / (a**2 * self.fct_yx(x, a, b))**2

def rayon_courbure(self, x, a, b):
    return abs((1 + self.fct_der1_yx(x, a, b)**2)**(3/2) / self.fct_der2_yz(x, a, b))
```

Nous avons choisi la valeur de $b = 8cm$, car c'est la distance à laquelle nous souhaitons que notre GoPiGo soit positionné par rapport au point O lorsqu'il effectue le stationnement. De plus, nous avons défini la valeur de a comme étant la distance entre l'emplacement de stationnement et le servo. Dans le schéma ci-dessous, nous avons pris la valeur de $a = 15cm$.



Voici la trajectoire souhaitée de notre véhicule. On essaie maintenant de l'approximer avec les rayons de courbures :



On remarque que les cercles approximent correctement la trajectoire elliptique en bleu.

Pour connaitre la distance à parcourir par chaque rayon de courbure on prend le périmètre de notre quart d'ellipse, on le divise par le nombre de rayon de courbure et on trouve dl.

Le périmètre d'une ellipse est approximé par la formule de Ramanujan :

$$P_{quart-ellipse} = \frac{\pi}{4}(a+b)(1+\frac{3h}{10+\sqrt{4-3h^2}}) \text{ ; avec } h = \frac{(a-b)^2}{(a+b)^2}$$

De plus on prend : $$dl = \frac{P_{quart-ellipse}}{Nombre\ de\ rayon\ de\ courbure}$$

En prenant ce dl on souhaite que sur chaque cercle d'approximation, la distance parcourue soit la même.

Ensuite, on sait que sur un cercle $dl = rd\theta$, avec r le rayon de courbure d'un arc de cercle. On a donc $d\theta = \frac{dl}{r}$.

On peut ensuite effectuer la trajectoire avec la fonction **gopigo.orbit(angle, rayon)** qui permet d'effectuer un mouvement circulaire selon un certains angle et un certain rayon.

```
def rangement_bataille(self):
    a = self.search_place()
    b = 8
    h = (a - b)**2 / (a + b)**2
    périmètre_quart_ellipse = pi / 4 * (a + b) * (1 + 3 * h / (10 + sqrt(4 - 3 * h ** 2)))
    x = [i for i in range(-a, 1, 2)]                                # On effectue un arc de cercle tous les 2cm
    xr = x[1:]
    xr.insert(0, -(a - 0.1))                                        # On ajoute le premier arc de cercle manuellement
    yr = [self.rayon_courbure(i, a, b) for i in xr]                 # Création de la liste avec les rayons de courbures
    dl = périmètre_quart_ellipse / len(yr)
    dtheta = [(dl / i) * (180 / pi) for i in yr]                    # Création de la liste avec les petits angles
    time.sleep(0.75)

    for i in range(len(yr)):                                        # On boucle sur toute la longeur de la liste
        time.sleep(0.20)
        self.gopigo.orbit(dtheta[i], yr[i])
```

Implémentation/validation

Après avoir mis en pratique notre code théorique, nous avons constaté que le robot présentait des écarts sur sa trajectoire, généralement de l'ordre de quelques centimètres. Pour corriger ces erreurs, nous avons effectué des ajustements empiriques. Voici les modifications que nous avons apportées à la fonction de stationnement en bataille, rangement_bataille() :

```
# Correction
self.gopigo.set_speed(100)
self.gopigo.turn_degrees(90 - sum(dtheta))                  # Pour finir la rotation à 90°
time.sleep(0.20)
self.servo.rotate_servo(90)
time.sleep(0.20)
obs = self.lidar.read()
self.gopigo.drive_cm(obs-3)                             # Pour que le robot s'arrete à 3cm avant la fin de la place
```

Le résultat est satisfaisant et répond à une partie de notre problématique. Voici une courte vidéo pour présenter notre implémentation :

Conclusion

Au cours de ce projet, nous avons été confrontés à une problématique exigeant la création de plusieurs modèles et la réalisation de recherches pour parvenir à une solution viable. Bien que le stationnement de véhicules puisse sembler une tâche anodine, il nécessite en réalité une mise en œuvre complexe pour minimiser les risques de collisions et d'accidents. De plus, il est impératif de prendre en compte tous les scénarios possibles et imaginables.

La modélisation que nous avons développée a démontré son efficacité dans le contexte du stationnement en bataille du côté passager. Notre robot est capable de suivre une trajectoire elliptique de manière précise, en évitant les obstacles, et ce, même lorsque la distance entre le véhicule et l'emplacement de stationnement varie. Cela représente un accomplissement significatif dans notre projet.