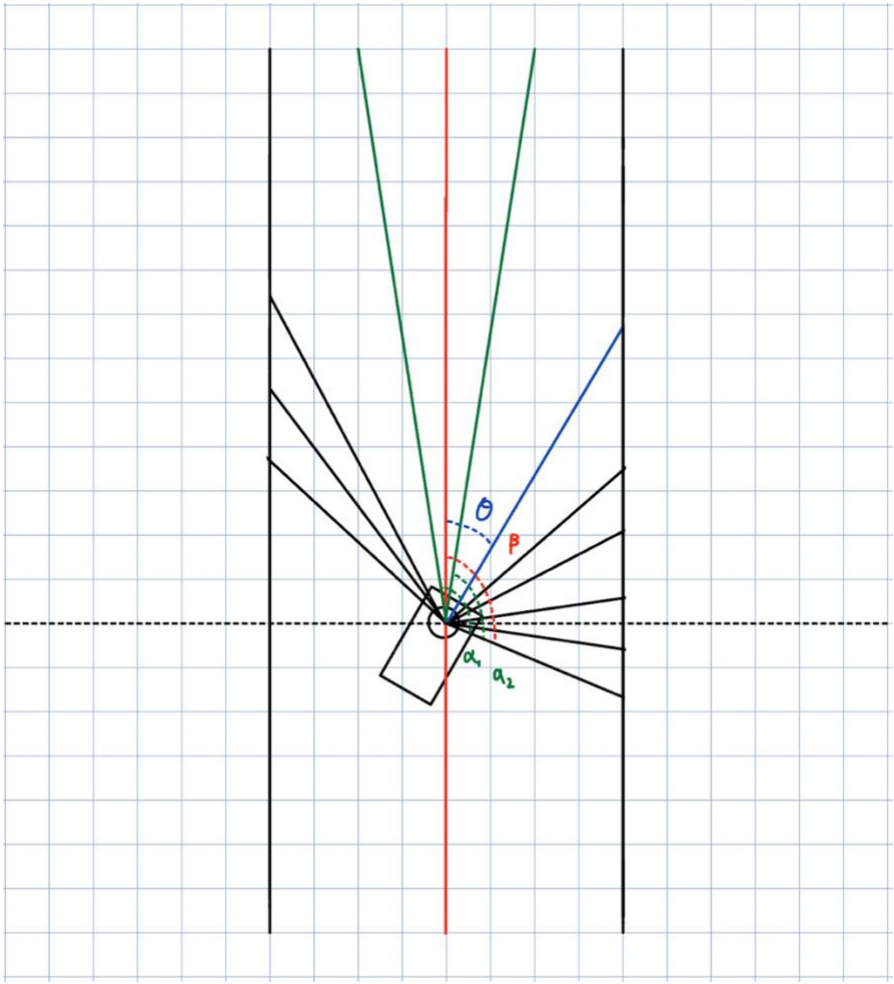
Présentation des nouvelles stratégies

Dans le présent rapport, les résultats des nouvelles stratégies seront présentés.

1) Stratégie 1 : Loi de direction pour bien aligner la voiture

La stratégie actuelle rencontre des problèmes lorsque la voiture roule en ligne droite. C'est pourquoi nous avons essayé de développer une nouvelle stratégie qui permettrait de maintenir la voiture parallèle à l'axe de la route.

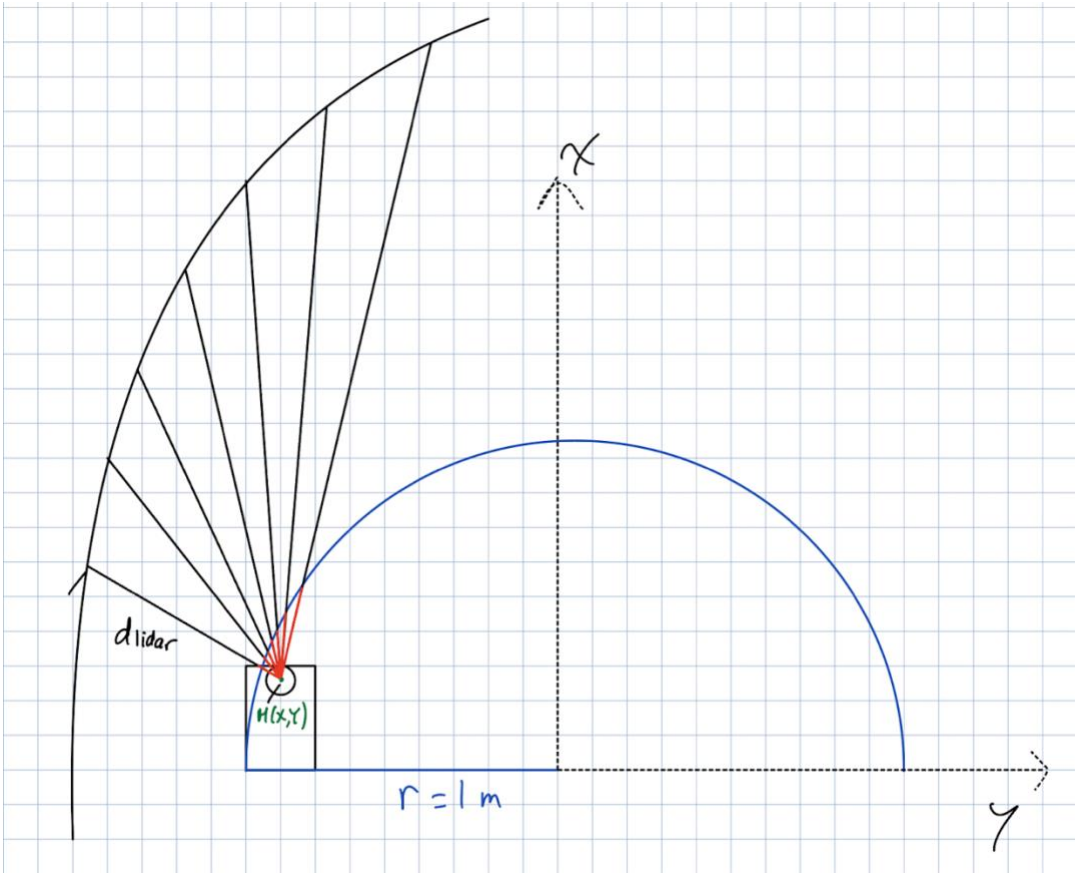


Les droites noire, bleue et rouge correspondent respectivement aux mesures Lidar, à l'axe de la voiture et à l'axe de la route. Pour aligner la voiture avec l'axe de la route, il suffit de la faire pivoter d'un angle θ. Le Lidar a une capacité limitée, ce qui peut causer des trous dans les mesures. Nous avons donc calculé la moyenne du trou (la moyenne des deux droites vertes) ce qui nous a permis d'obtenir la direction de l'axe de la route représentée par la droite rouge. Cette stratégie a été testée sur un simulateur et le code utilisé pour la simulation est présenté en annexe A. Cependant, nous avons observé que la voiture ne se comportait pas

correctement lors de la simulation. Elle heurtait rapidement les murs ou ne changeait pas de direction dans certains cas, ce qui suggère un problème. Nous n'avons pas été en mesure de déterminer la cause exacte de ce problème. Cependant, nous soupçonnons que cela pourrait être dû à la performance limitée du Raspberry Pi utilisé ou à une erreur d'implémentation du code.

2) Stratégie 2 : Loi de direction utilisant le rayon de braquage de la voiture

La deuxième stratégie consiste à utiliser la distance maximale entre le rayon de braquage de la voiture et le mur, plutôt que la distance maximale fournie par le Lidar. Le rayon de braquage r a été mesuré par une équipe de l'année précédente.



Nous avons placé l'origine du repère au centre du cercle de braquage. Cependant, lors des simulations, nous avons constaté que le comportement de la voiture était loin de ce que nous avions anticipé. Encore une fois, il est possible que ce problème soit dû à la performance limitée du Raspberry Pi ou à une erreur d'implémentation du code. Le code utilisé pour la simulation se trouve en annexe B.

Annexe A :

```
# Copyright 1996-2022 Cyberbotics Ltd.
#
# Controle de la voiture TT-02 simulateur CoVAPSy pour Webots 2022b
# Inspiré de vehicle_driver_altino controller
# Kévin Hoarau, Anthony Juton, Bastien Lhopitallier, Martin Taynaud
# janvier 2023

## CONST

import math
from math import *
from math import sin, cos, radians, sqrt, pi, atan

vmin = 0
vmax = 28
d = 2.3

## FILTRAGE

def filtrage(map, largeur_convolution):
    n=len(map)
    mapt=[0]*n
    for i in range(n):
        for k in range(-largeur_convolution,largeur_convolution):
            mapt[i]+=map[abs(i+k)%n]
            mapt[i]/=(2*largeur_convolution+1)
    return mapt


from vehicle import Driver
from controller import Lidar

driver = Driver()

basicTimeStep = int(driver.getBasicTimeStep())
sensorTimeStep = 4 * basicTimeStep
```

```
#Lidar
lidar = Lidar("lidar")
lidar.enable(sensorTimeStep)
lidar.enablePointCloud()

keyboard = driver.getKeyboard()
keyboard.enable(sensorTimeStep)

# vitesse en km/h
speed = 0
maxSpeed = 28 #km/h

# angle de la direction
angle = 0
maxangle = 0.28 #rad (étrange, la voiture est défini pour une limite à 0.31 rad...

# mise a zéro de la vitesse et de la direction
driver.setSteeringAngle(angle)
driver.setCruisingSpeed(speed)
# mode manuel et mode auto desactive
modeManuel = False
modeAuto = True

print("cliquer sur la vue 3D pour commencer")
print("m pour mode manuel, a pour mode auto, n pour stop, l pour affichage données lidar")
print("en mode manuel utiliser les flèches pour accélérer, freiner et diriger")

while driver.step() != -1:

    speed = driver.getTargetCruisingSpeed()

    while True:
        #acquisition des donnees du lidar
        donnees_lidar = lidar.getRangeImage()

        # recuperation de la touche clavier
        currentKey = keyboard.getKey()
        if currentKey == -1:
            break
        if currentKey == ord('m') or currentKey == ord('M'):
```

```
        if not modeManuel:
            modeManuel = True
            modeAuto = False
            print("------------Mode Manuel Activé----------")
    elif currentKey == ord('n') or currentKey == ord('N'):
        if modeManuel or modeAuto :
            modeManuel = False
            modeAuto = False
            print("--------Modes Manuel et Auto Désactivé-------")
    elif currentKey == ord('a') or currentKey == ord('A'):
        if not modeAuto :
            modeAuto = True
            modeManuel = False
            print("------------Mode Auto Activé----------------")
    elif currentKey == ord('l') or currentKey == ord('L'):
            print("-----donnees du lidar en metres sens horaire de -99° à +99° au pas de 2°-----")
            for i in range(len(donnees_lidar)) :
                print(f"{donnees_lidar[i]:.3f}   ", end='')
                if (i+1)%10 == 0 :
                    print()


    # Controle en mode manuel
    if modeManuel:
        if currentKey == keyboard.UP:
            speed += 0.2
        elif currentKey == keyboard.DOWN:
            speed -= 0.2
        elif currentKey == keyboard.LEFT:
            angle -= 0.04
        elif currentKey == keyboard.RIGHT:
            angle += 0.04

if not modeManuel and not modeAuto:
    speed = 0
    angle = 0

if modeAuto:
################## LOI DE DIRECTION ###################
```

```
##############################
        cone_detect = 50
        mapt=filtrage(donnees_lidar, 5)
        print(mapt)
        for i in range(len(mapt)):
            c = 10

            d1 = mapt[i]

            if i+c > len(mapt):
                c = len(mapt) - i

            d2 = mapt[i+c-1]


            y1 = d1*sin(i)
            x1 = d1*cos(i)
            y2 = d2*sin(i)
            x2 = d2*cos(i)
            if x2-x1 == 0:
                x1 = 1
            #print(x1, y1, x2, y2)
            pente = (y2 - y1)/(x2- x1)

                #ligne droite
            if pente > 100:
                for j in range(-cone_detect, cone_detect):
                    d = mapt[j]
                    # L'unite de la distance ?
                    if j < 0 and d > 3:
                        alpha1 = j
                    elif j > 0 and d > 3:
                        alpha2 = j
                angle_cible = (alpha1 + alpha2)/2
            #virage
            else:
                #calcul du rayon interieur de la piste
                d_gauche = mapt[0]
                d_droite = mapt[len(mapt)-1]
                largeur_piste = d_gauche + d_droite
```

```
            #calcul du rayon de braquage
            t = sqrt(d2**2 + d1**2 - 2*d2*d1*cos(c))
            r_ext = (2/pi)*t
            r_int = r_ext - largeur_piste
            r_cible = (r_ext + r_int)/2
            #calcul de l'angle de braquage
            d_roues = 0.25 # cm
            if r_cible == 0:
                r_cible = 1
            angle_cible = atan(d_roues/r_cible)

    print(mapt[::10],angle_cible)

    if abs(angle_cible)<20:
        angle_consigne=0
    elif abs(angle_cible)<55:
        angle_consigne=min(10,max(-10,angle_cible))*3.14/180
    else:
        angle_consigne=min(22,max(-22,angle_cible))*3.14/180

    if angle_consigne > maxangle:
        angle_consigne = maxangle
    elif angle_consigne < -maxangle:
        angle_consigne = -maxangle
    ####################  LOI DE VITESSE  ####################
    speed=vmin+(vmax-vmin)*(1-math.exp(-mapt[44]*3/d))
    ################### FIN CONSIGNE VITESSE ###################
    if speed > maxSpeed:
        speed = maxSpeed
    elif speed < -1 * maxSpeed:
        speed = -1 * maxSpeed

# clamp speed and angle to max values
print(len(donnees_lidar))

driver.setCruisingSpeed(speed)
driver.setSteeringAngle(angle_consigne)
```

Annexe B :

```
# Copyright 1996-2022 Cyberbotics Ltd.
#
# Controle de la voiture TT-02 simulateur CoVAPSy pour Webots 2022b
# Inspiré de vehicle_driver_altino controller
# Kévin Hoarau, Anthony Juton, Bastien Lhopitallier, Martin Taynaud
# janvier 2023

## CONST

import math
from math import *
from math import sin, cos, radians, sqrt, pi, atan

vmin = 0
vmax = 28
d = 2.3

## FILTRAGE

def filtrage(map, largeur_convolution):
    n=len(map)
    mapt=[0]*n
    for i in range(n):
        for k in range(-largeur_convolution,largeur_convolution):
            mapt[i]+=map[abs(i+k)%n]
            mapt[i]/=(2*largeur_convolution+1)
    return mapt


from vehicle import Driver
from controller import Lidar

driver = Driver()

basicTimeStep = int(driver.getBasicTimeStep())
sensorTimeStep = 4 * basicTimeStep

#Lidar
lidar = Lidar("lidar")
```

```
lidar.enable(sensorTimeStep)
lidar.enablePointCloud()

keyboard = driver.getKeyboard()
keyboard.enable(sensorTimeStep)

# vitesse en km/h
speed = 0
maxSpeed = 28 #km/h

# angle de la direction
angle = 0
maxangle = 0.28 #rad (étrange, la voiture est défini pour une limite à 0.31 rad...

# mise a zéro de la vitesse et de la direction
driver.setSteeringAngle(angle)
driver.setCruisingSpeed(speed)
# mode manuel et mode auto desactive
modeManuel = False
modeAuto = True

print("cliquer sur la vue 3D pour commencer")
print("m pour mode manuel, a pour mode auto, n pour stop, l pour affichage données lidar")
print("en mode manuel utiliser les flèches pour accélérer, freiner et diriger")

while driver.step() != -1:

    speed = driver.getTargetCruisingSpeed()

    while True:
        #acquisition des donnees du lidar
        donnees_lidar = lidar.getRangeImage()

        # recuperation de la touche clavier
        currentKey = keyboard.getKey()
        if currentKey == -1:
            break
        if currentKey == ord('m') or currentKey == ord('M'):
            if not modeManuel:
                modeManuel = True
```

```
            modeAuto = False
            print("------------Mode Manuel Activé---------------")
        elif currentKey == ord('n') or currentKey == ord('N'):
            if modeManuel or modeAuto :
                modeManuel = False
                modeAuto = False
                print("--------Modes Manuel et Auto Désactivé-------")
        elif currentKey == ord('a') or currentKey == ord('A'):
            if not modeAuto :
                modeAuto = True
                modeManuel = False
                print("------------Mode Auto Activé----------------")
        elif currentKey == ord('l') or currentKey == ord('L'):
                print("-----donnees du lidar en metres sens horaire de -99° à +99° au pas de 2°-----")
                for i in range(len(donnees_lidar)) :
                    print(f"{donnees_lidar[i]:.3f}   ", end='')
                    if (i+1)%10 == 0 :
                        print()
                print()



        # Controle en mode manuel
        if modeManuel:
            if currentKey == keyboard.UP:
                speed += 0.2
            elif currentKey == keyboard.DOWN:
                speed -= 0.2
            elif currentKey == keyboard.LEFT:
                angle -= 0.04
            elif currentKey == keyboard.RIGHT:
                angle += 0.04

    if not modeManuel and not modeAuto:
        speed = 0
        angle = 0

    if modeAuto:
    ################## LOI DE DIRECTION ####################
```

```
##############################
        cone_detect = 50
        mapt=filtrage(donnees_lidar, 5)
        print(mapt)
        for i in range(len(mapt)):
            c = 10

            d1 = mapt[i]

            if i+c > len(mapt):
                c = len(mapt) - i

            d2 = mapt[i+c-1]


            y1 = d1*sin(i)
            x1 = d1*cos(i)
            y2 = d2*sin(i)
            x2 = d2*cos(i)
            if x2-x1 == 0:
                x1 = 1
            #print(x1, y1, x2, y2)
            pente = (y2 - y1)/(x2- x1)

                #ligne droite
            if pente > 100:
                for j in range(-cone_detect, cone_detect):
                    d = mapt[j]
                    # L'unite de la distance ?
                    if j < 0 and d > 3:
                        alpha1 = j
                    elif j > 0 and d > 3:
                        alpha2 = j
                angle_cible = (alpha1 + alpha2)/2
            #virage
            else:
                #calcul du rayon interieur de la piste
                d_gauche = mapt[0]
                d_droite = mapt[len(mapt)-1]
                largeur_piste = d_gauche + d_droite
```

```
            #calcul du rayon de braquage
            t = sqrt(d2**2 + d1**2 - 2*d2*d1*cos(c))
            r_ext = (2/pi)*t
            r_int = r_ext - largeur_piste
            r_cible = (r_ext + r_int)/2
            #calcul de l'angle de braquage
            d_roues = 0.25 # cm
            if r_cible == 0:
                r_cible = 1
            angle_cible = atan(d_roues/r_cible)

    print(mapt[::10],angle_cible)

    if abs(angle_cible)<20:
        angle_consigne=0
    elif abs(angle_cible)<55:
        angle_consigne=min(10,max(-10,angle_cible))*3.14/180
    else:
        angle_consigne=min(22,max(-22,angle_cible))*3.14/180

    if angle_consigne > maxangle:
        angle_consigne = maxangle
    elif angle_consigne < -maxangle:
        angle_consigne = -maxangle
    ####################  LOI DE VITESSE  ####################
    speed=vmin+(vmax-vmin)*(1-math.exp(-mapt[44]*3/d))
    ################### FIN CONSIGNE VITESSE ###################
    if speed > maxSpeed:
        speed = maxSpeed
    elif speed < -1 * maxSpeed:
        speed = -1 * maxSpeed

# clamp speed and angle to max values
print(len(donnees_lidar))

driver.setCruisingSpeed(speed)
driver.setSteeringAngle(angle_consigne)
```