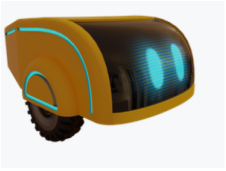
Challenge

Choisir un projet par exemple ici « Suiveur de ligne » - Line Follower

Cliquer sur « Fork » puis « Open Project »



Line Follower Competition Prep Track

Open Project Fork

L'environnement (figure2) qui apparait est le suivant, une arène ou l'on distingue le petit robot en jaune et la ligne du circuit à suivre (en vert). Les monuments ne servent que de décoration, l'arène est le modèle 3D du Colisée et la basilique, St Pierre de Rome.

Figure2

Le robot (figure 3) est un classique robot suiveur de ligne à 2 roues motrices indépendantes avec roue folle à l'arrière. A noter que le pilotage, lui n'est pas classique. La stratégie consiste à piloter l'avance (robot.move()) et l'orientation (Robot.rotate())

Il dispose d'une caméra RGB pour la détection de ligne.

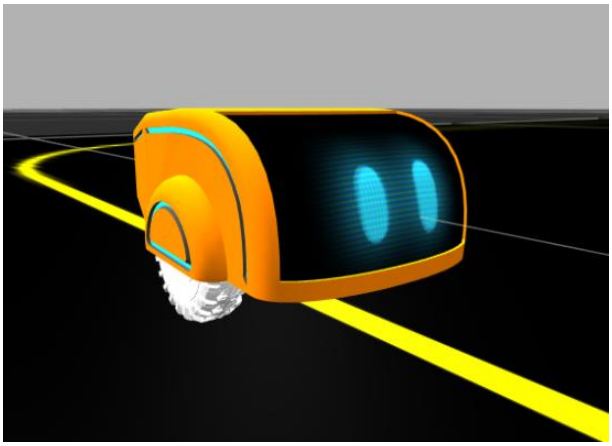


Figure 3

Vision de la caméra

Le signal reçu est codé sur une grille de 32 lignes, la caméra doit idéalement se trouver au milieu (avec la luminosité maximale) pour suivre la ligne.

Figure 4

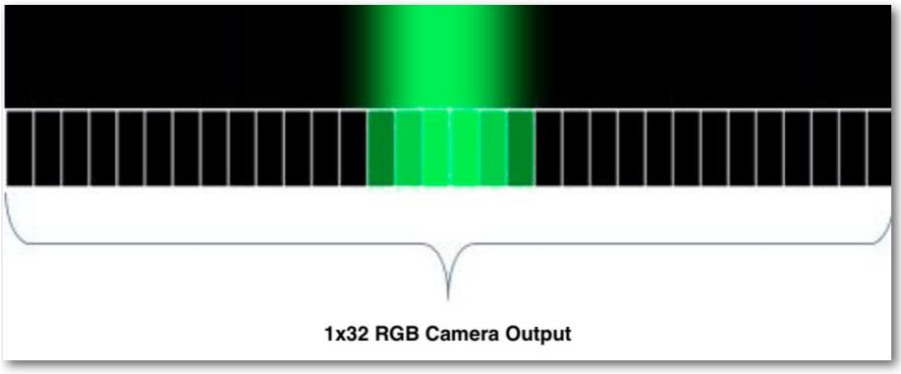


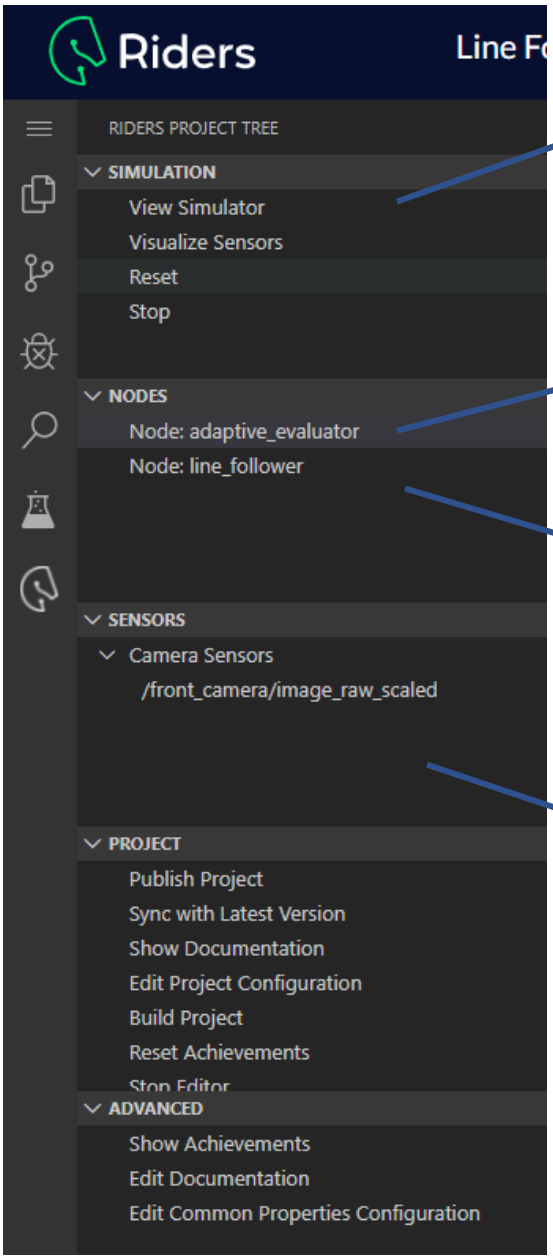
Figure5

Dans ce cas ci-dessous, il s'agit des lignes 25 et 26 ou la sortie du capteur est maximale. Le robot se trouve alors trop à gauche du tracé. Une correction avec l'opérateur « robot.rotate() » est nécessaire.

[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 68.632, 151.213, 230.732, 236.589, 236.589, 232.877, 151.8169, 73.92, 0.0, 0.0, 0.0]

Environnement de programmation

Une fois que le projet est ouvert, l'interface suivante apparait.



Une fois cliquer sur « Start » la simulation démarre et vous accéder à ce panneau de contrôle.

Dans « Nodes » cliquer sur « Node : « adaptative_evaluator » le programme ci-dessous (figure 6) apparait. Il s'agit du programme relatif à la plateforme et au paramétrage du robot

Dans « Nodes » cliquer sur « Node : « line_evaluator » le programme proprement dit de suivi de ligne est à compléter. Voir exemple figure 6 (programme de base qui permet d'animer le robot)

En cliquant sur la caméra on voit le positionnement de la ligne – Figure 4

```python
#! /usr/bin/env python
from __future__ import division

import rospy
import re
import math
import json
import datetime
import os
import requests
import subprocess

from geometry_msgs.msg import Point
from gazebo_msgs.srv import GetModelProperties
from gazebo_msgs.srv import GetModelState
from gazebo_msgs.srv import GetWorldProperties

from std_msgs.msg import String
from line_follower_evaluator.msg import EvaluatorMetric

from tf.transformations import quaternion_from_euler, euler_from_quaternion
from angles import normalize_angle, shortest_angular_distance

from line_follower_evaluator.tile import Tile


class AdaptiveEvaluatorNode:

    def __init__(self):

        self.nodeName = 'adaptive_evaluator'
        self.robotName = 'robot1'
        self.git_dir = '/workspace/src/.git'

        self.start_x = 0.5
        self.start_y = -3.5
```

Figure 6

Les commandes du robot

- `robot.is_ok()` - True while the simulation is running.
- `robot.move(v)` - Set robot speed v [meter/sec].
- `robot.rotate(ω)` - Set robot angular velocity ω [radian/sec]. Positive ω is counter-clockwise (CCW). Negative ω is clockwise (CW).
- `robot.image_data()` - Used to the get the image from the camera.
- `robot.get_sensor_data()` - Used to read the pixels of the camera image.
- `robot.x` - Holds the robot's x position.
- `robot.y` - Holds the robot's y position.
- `robot.yaw` - Holds the robot's rotation in radians.
- `robot.vel_x` - Holds the robot's velocity in the x direction.
- `robot.vel_y` - Holds the robot's velocity in the y direction.

Voici un exemple de programme de base pour débuter, à modifier pour obtenir le meilleur score. Les commentaires permettent une analyse de l'algorithme nécessaire au suivi de ligne. Il est aussi fourni en vers Python.

Contrairement à du suivi de ligne classique ou l'on pilote chaque moteur en vitesse, ici nous avons 2 fonctions. Une pour l'avance du robot ; robot.move(valeur_vitesse) et une pour la rotation du robot ; robot.rotate(valeur_vitesse_angulaire).

```python
from __future__ import division

import rospy
import math
import time

from rosrider_lib.rosrider import ROSRider

robot = ROSRider('rosrider')

#Your code starts here. Use while.robot.is_ok() as your loop statement:
while robot.is_ok():
    # Obtenir l'image de la ligne
    sensor_row = robot.get_sensor_data()# Le capteur dispose de 32 ligne de lecture, len (sensor_row) ci dessous
    # Chercher la ligne
    max = 0# Initialisation de la variable max
    index = 0
    for i in range(0, len(sensor_row)):# Une  boucle qui permet de vérifier l'état de chaque ligne du capteur et de trouver la valeur maxi
        if sensor_row[i] > max:# Au départ on compare si la valeur du capteur est supérieur à zéro, si c'est le cas on affecte cette valeur à max
            max = sensor_row[i] # Cela permet de déterminer la valeur maxi du capteur, normalement 255 en pleine lumière
            index = i # on affecte alors la valeur du numéro de capteur à index, idéalement 16 si c'est la ligne du milieu
    # Follow the line
    error = (len(sensor_row) / 2) - index #© Calcul de l'erreur 32/2 - valeur de l'index - si la ligne se trouve sur le 5; 16-5=11
    gain = 0.1 # une amplification de l'erreur
    command = gain*error# On corrige la commande de rotation, dans notre exemple 11*0.1 = 1.1
    robot.move(0.3)# Vitesse de déplacement du robot en m/s
    robot.rotate(command)# Commande de la rotation du robot, ici 1.1 rad/s
```

Figure 7

En cliquant sur « start » la simulation démarre, une fois le parcours réalisé on voit dans le panneau de contrôle inférieur le score et le temps de parcours. L'objectif étant d'améliorer le temps de parcours pour monter dans le classement.

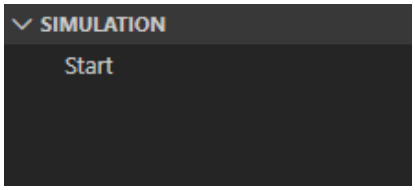


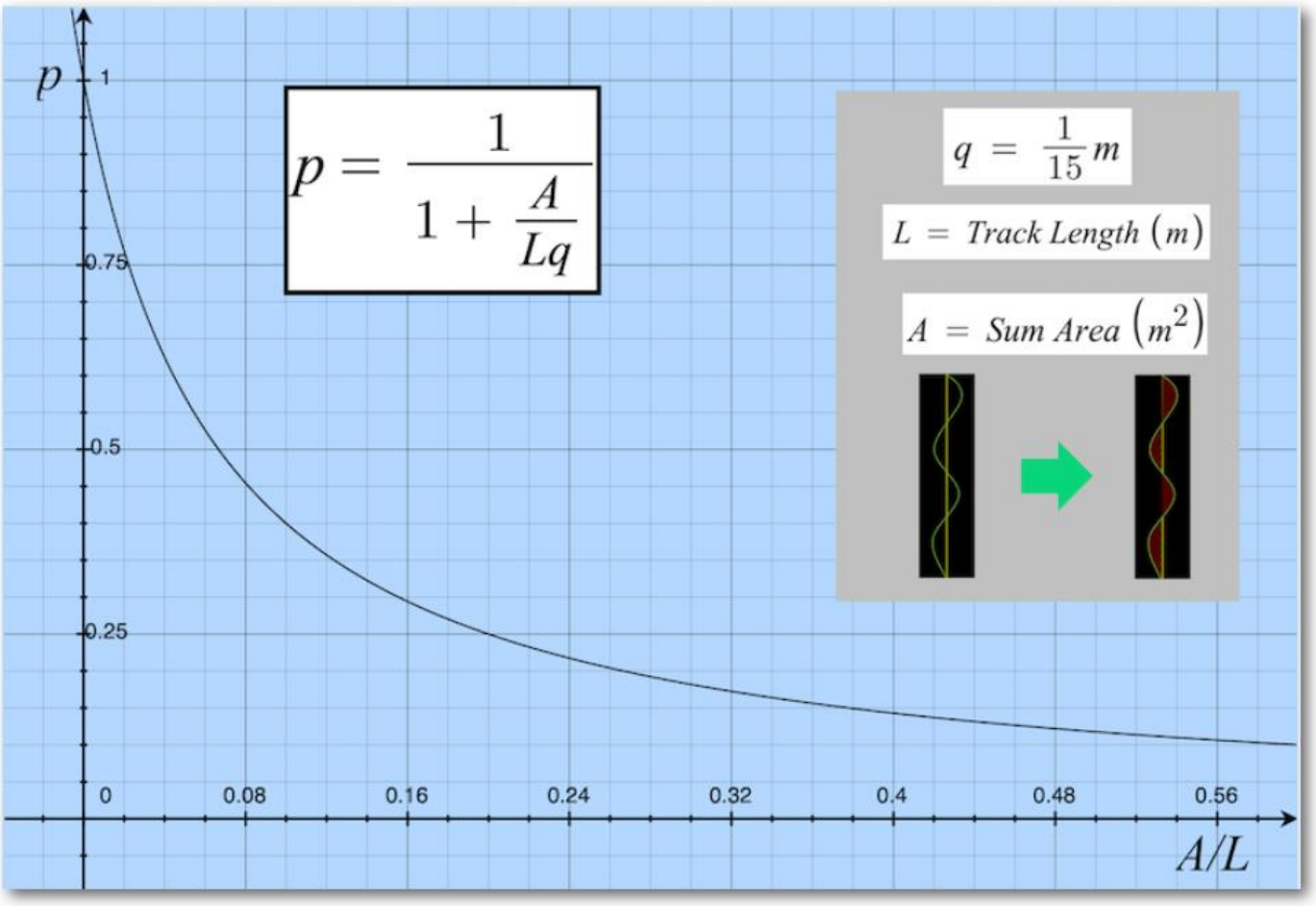
Figure5

Calcul du score

Le calcul du score est basé sur une formule mathématique ou L est la distance du parcours (constante), t le temps de parcours et p un facteur de performance variant de 0 à 1.

$$Score = \frac{L}{t} x\, p$$

Le facteur de performance est indexé sur les écarts avec la ligne à suivre. La somme des aires A « Sum Area » (m^2) correspondant à la déviation est aussi prise en compte selon la formule ci-dessous. Un autre facteur, q, relatif à la déviation qui induit une diminution du score de ½ pour un décalage de 1/15 de m.



Une fois que le challenge est achevé, notre profil apparaît dans l'historique

Project History

latest — laurent_gimazane — February 3, 2022 12:02 PM

v10 — Maker fair branches — zaphara — December 3, 2020 12:21 AM

Scénarii possibles

Scénario1 – Déterminer la vitesse maximale de déplacement avec le même gain de correction

On modifie simplement la vitesse du robot de manière progressive afin de trouver la limite acceptable qui permet de suivre la ligne.

Voici les résultats obtenus.

Robot.move(0.25) au lieu de 0.3

Temps plus long, score plus faible

Robot.move(0.6) au lieu de 0.3

Fail, le robot ne réussit pas à prendre le virage

Robot.move(0.35) au lieu de 0.3

Temps plus court, score plus élevé

Robot.move(0.5) au lieu de 0.3

Temps plus court, score plus élevé

Robot.move(0.58) au lieu de 0.3

Temps plus court, score plus élevé

Robot.move(0.59) au lieu de 0.3

Vitesse la plus rapide acceptable

Scénario2 – Ajuster le gain de correction, pour améliorer le score avec cette vitesse maximale

Robot.move(0.59) + Gain = 0.08 Fail

Robot.move(0.59) + Gain = 0.09

Un légère augmentation du temps et une baisse du score, le sum area a augmenté

Robot.move(0.59) + Gain = 0.11

Amélioration du temps mais le sum area augmente aussi donc le score baisse

Scénario3 – Vérifier que ce gain permet de corriger les « sorties de route » à des vitesses supérieures

Robot.move(0.6) + Gain = 0.11

disqualification reason :	**race finished** : True	**race started** : True	**score** : 0.540530777295
sum area : 0.402500294226	**time** : 79.648		

Baisse du temps, meilleur score

Robot.move(0.65) + Gain = 0.11

disqualification reason :	**race finished** : True	**race started** : True	**score** : 0.59504604498
sum area : 0.316714066834	**time** : 74.102		

Robot.move(0.7) + Gain = 0.12

disqualification reason :	**race finished** : True	**race started** : True	**score** : 0.632185768033
sum area : 0.363867733486	**time** : 68.833		

Robot.move(0.72) + Gain = 0.13

disqualification reason :	**race finished** : True	**race started** : True	**score** : 0.639441673615
sum area : 0.430966186791	**time** : 66.804		

Robot.move(0.75) + Gain = 0.14

disqualification reason :	**race finished** : True	**race started** : True	**score** : 0.652704263106
sum area : 0.496559816428	**time** : 64.294		

Robot.move(0.75) + Gain = 0.14

disqualification reason :	**race finished** : True	**race started** : True	**score** : 0.67093759855
sum area : 0.481185664484	**time** : 62.806		

Attention valeur limite de la vitesse, le sum area augmente

Scénario4 – Changer de correcteur avec un PID Robot.move(0.75)

```python
from __future__ import division

import rospy
import math
import time

from rosrider_lib.rosrider import ROSRider

robot = ROSRider('rosrider')

integralArea = 0.0
previousError = 0.0

#Your code starts here. Use while.robot.is_ok() as your loop statement:
while (robot.is_ok()):
    # Obtenir l'image de la ligne
    sensor_row = robot.get_sensor_data()
    # Chercher la ligne
    max = 0
    index = 0

    for i in range(0, len(sensor_row)):
        if sensor_row[i] > max:
            max = sensor_row[i]
            index = i
    # Suivre la ligne
    error = (len(sensor_row) / 2) - index
    Kp = 0.14 #
    Ki = 0.0 #
    Kd = 0.0 #
    integralArea += error
    command = Kp * error + Ki * integralArea + Kd * (previousError - error)
    previousError = error
    robot.move(0.77)
    robot.rotate(command)
```

Kp = 0.14
Ki = 0.0
Kd = 0.0

disqualification reason :	**race finished** : True	**race started** : True	**score** : 0.653373097071
sum area : 0.485988467912	**time** : 64.411		

Kp = 0.14
Ki = 0.0
Kd = 0.01

disqualification reason :	**race finished** : True	**race started** : True	**score** : 0.656681825147
sum area : 0.476938679913	**time** : 64.243		

Kp = 0.13
Ki = 0.0
Kd = 0.01

disqualification reason :	**race finished** : True	**race started** : True	**score** : 0.651578141473
sum area : 0.479469818326	**time** : 64.702		

Kp = 0.15
Ki = 0.0
Kd = 0.012

disqualification reason :	**race finished** : True	**race started** : True	**score** : 0.64997321848
sum area : 0.524934753047	**time** : 64.076		

Kp = 0.15
Ki = 0.0
Kd = 0.008

disqualification reason :	**race finished** : True	**race started** : True	**score** : 0.651807833771
sum area : 0.513904278622	**time** : 64.084		

Kp = 0.13
Ki = 0.0
Kd = -0.009

disqualification reason :	**race finished** : True	**race started** : True	**score** : 0.668528175229
sum area : 0.400667624184	**time** : 64.431		

Kp = 0.13 pour Robot.move(0.78)
Ki = 0.0
Kd = -0.009

disqualification reason :	**race finished** : True	**race started** : True	**score** : 0.686663660089
sum area : 0.439389084402	**time** : 62.067		

Kp = 0.17 pour Robot.move(0.8)
Ki = 0.0
Kd = -0.011

disqualification reason :	**race finished** : True	**race started** : True	**score** : 0.698389218585
sum area : 0.478806981858	**time** : 60.376		

Kp = 0.17 pour Robot.move(0.8)
Ki = 0.005
Kd = -0.011

disqualification reason :	**race finished** : True	**race started** : True	**score** : 0.621119536871
sum area : 0.937329419447	**time** : 60.414		

Kp = 0.18 pour Robot.move(0.8)
Ki = 0.001
Kd = -0.011

disqualification reason :	**race finished** : True	**race started** : True	**score** : 0.6898885824
sum area : 0.538392960646	**time** : 60.153		