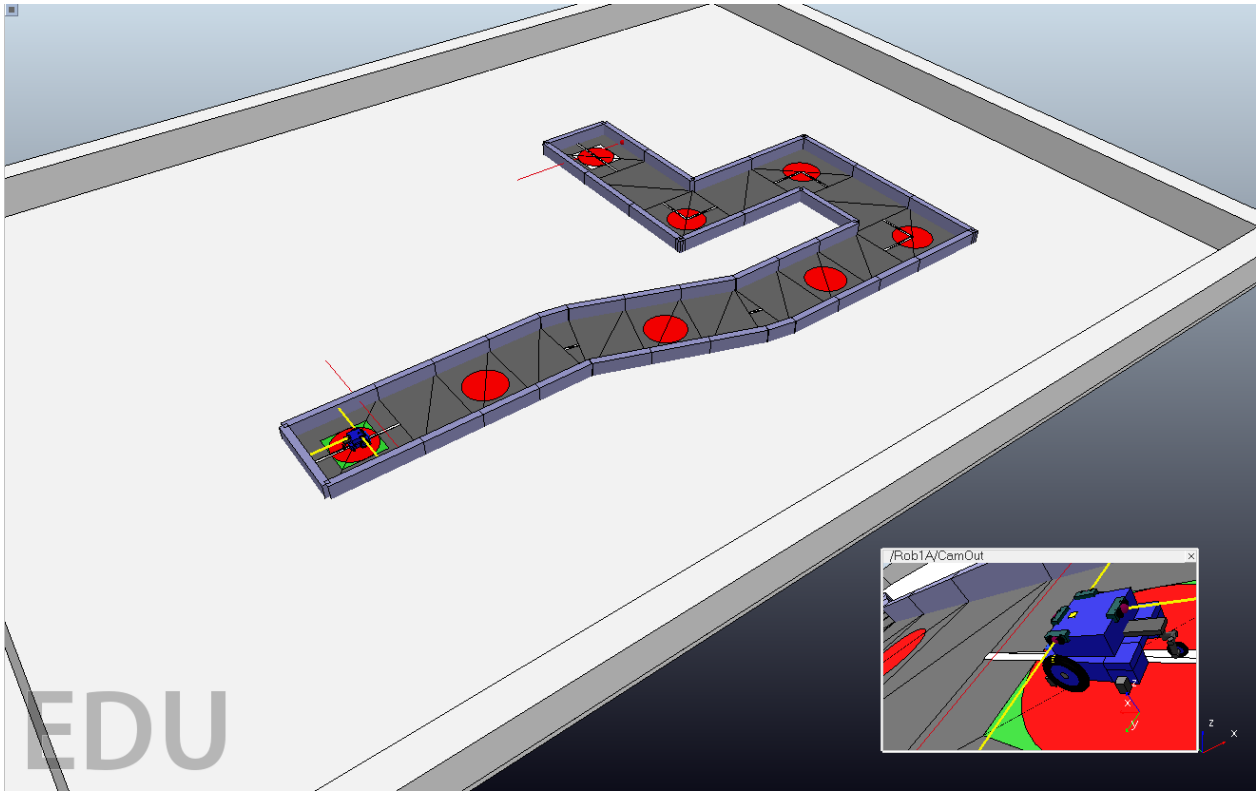


FISE 1A - UE 2.2 - Tutoriel TD 2

benoit.zerr@ensta-bretagne.org

Tutoriel TD2

Bien débuter le challenge qualif2

- Réaliser un suivi de la ligne blanche centrale et s'arrêter face au mur frontal
- Tester les capteurs de réflectivité
- Définir un seuil de déclenchement (threshold)
- Définir la commande de vitesse en fonction du déclenchement
- Utiliser la télémétrie

Tutoriel TD2

Créer un nouveau programme

On peut créer par exemple : q2_test.py

Importer les modules python utiles :

```
import rob1a_v03 as rob1a  # get the robot code
import control  # robot control functions
import filt # sensors filtering functions
import numpy as np
import time
```

Tutoriel TD2

Créer un nouveau programme

- Creation des objets utiles pour utiliser le robot **rb** et les boucles de contrôle **ctrl**. Attention à ne créer qu'un seul object robot **rb**, la version de votre simulateur ne peut gérer qu'un seul robot. Si il y a plusieurs robots, ils ne peuvent pas se connecter au simulateur et l'erreur suivante apparaît :

```
File "zmq/backend/cython/socket.pyx", line 568, in zmq.backend.cython.socket.Socket.bind
File "zmq/backend/cython/checkrc.pxd", line 28, in zmq.backend.cython.checkrc._check_rc
zmq.error.ZMQError: Address already in use
```

- Définition du pseudo (nickname) qui vous servira à reconnaître votre performance dans les tableaux de résultats à partir du 1^{er} mars 2023.
Si non défini un pseudo est créé par défaut (ex pour Alain PROST cela donnerait : AlainP..ST)

- Code Python :

```python
if __name__ == "__main__":
    rb = rob1a.Rob1A()    # create a robot (instance of Rob1A class)
    rb.set_pseudo("myAwesomePseudo") # you can define your pseudo here
    ctrl = control.RobotControl() # create a robot controller
```

Tutoriel TD2

Télémétrie : "log file"

- On peut valider ou non la télémétrie avec rb.set_log(log)
- log = 0 pas de télémétrie
- log = 1 la télémétrie fournit à la fin de l'exécution du programme Python un fichier rob1a.log dans le dossier ou se trouve la scène (ici qualif2A.ttt)
- le fichier "rob1a.log" est un fichier au format .csv utilisable sous OpenOffice ou Excel
- l'affichage des données peut être réalisé en Python en modifiant "plot_mission.py"
- Code Python :

```
# log data to analyse sensors
rb.set_log(1)    # do not log data (set to 1 for logging data)
```

Tutoriel TD2

Test des capteurs de refléctivité

- Comprendre le fonctionnement des capteurs de réflectivité pour la détection de la ligne blanche
- Mesurer le bruit
- Ecrire une nouvelle boucle de contrôle : avancer en ligne droite pendant 12 secondes : fonction **follow_center_line_v0()**
- Code Python

```
# follow the center line until duration_max is reached
spd_lin = 400
duration_max = 12.0 # set a max duration of 12 seconds
ctrl.follow_center_line_v0 (rb,spd_lin,duration_max)
```

Tutoriel TD2

Une fin éléguante ...

- Pour bien terminer écrire la commande **rb.full_end()** à la toute fin du programme
- Cette commande ne doit apparaître qu'une seule fois à la fin du programme principale
- Cette commande garantit une bonne évaluation automatique
- Note : il est aussi possible d'interrompre "proprement" le programme avec les touches simultanées "Control" + "C" ou avec le bouton stop (carré rouge) de PyCharm
- Code Python

```
rb.full_end() # clean end of simulation
```

Tutoriel TD2

Récapitulatif de la mission d'acquisition des capteurs de réflectivité

```
import rob1a_v03 as rob1a  # get the robot code
import control  # robot control functions
import filt # sensors filtering functions
import numpy as np
import time

if __name__ == "__main__":
    rb = rob1a.Rob1A()    # create a robot (instance of Rob1A class)
    rb.set_pseudo ("myAwesomePseudo") # you can define your pseudo here
    ctrl = control.RobotControl() # create a robot controller

    # log data to analyse sensors
    rb.set_log(1)    # do not log data (set to 1 for logging data)

    # follow the center line until duration_max is reached
    spd_lin = 400
    duration_max = 12.0 # set a max duration of 12 seconds
    ctrl.follow_center_line_v0 (rb,spd_lin,duration_max)

    rb.full_end() # clean end of simulation
```

Tutoriel TD2

Boucle de contrôle

- Dans le module **control.py**, ajouter la boucle de contrôle **follow_center_line_v0()**
- Décrire la fonction pour simplifier son utilisation future
- Code Python

```
def follow_center_line_v0 (self,rb,spd_lin,duration_max):
    """
    Follow the white center line using IR reflective sensors.
    Stop at 25 cm of a front obstacle (end wall).

    Input parameters :
    rb : robot
    spd_lin : linear speed
    duration_max : maximum duration of the move

    Output parameters :
    None
    """
```

Tutoriel TD2

Boucle de contrôle

- Code Python de la fonction **follow_center_line_v0()**

```
loop_iter_time = 0.1 # control at 10 Hz (10 commands/second)
t_start = time.time()
while True:
    t0loop = time.time()
    if (time.time() - t_start) > duration_max:
        break # max time reached , escape the loop ...
    # acquire sensors
    rl,rm,rr = rb.get_centerline_sensors()
    # set actuators
    rb.set_speed(spd_lin,spd_lin)
    # end of loop
    t1loop = time.time()
    dt_sleep = loop_iter_time - (t1loop - t0loop)
    if (dt_sleep > 0):
        time.sleep(dt_sleep) # wait to have perfect loop duration
    else:
        print ("too much computation in this loop")
        # increase loop_iter_time or simplify computation ...
# stop the robot
rb.stop()
```

Tutoriel TD2

"log file" : fichier de télémétrie

- le fichier de télémétrie enregistre les mesures et les commandes à chaque instant de la mission
- il permet ensuite d'analyser la mission
- le fichier de télémétrie se nomme "rob1a.log"
- le fichier "rob1a.log" se trouve dans le même dossier que le challenge (ex. dossier qualif2)
- à chaque pas de temps, 18 données (valeurs numériques) sont enregistrées
- les données peuvent être visualisées et analysées en modifiant "plot_mission.py"

Tutoriel TD2

"log file" : les 18 données enregistrées

- définition des 18 données enregistrées

```
#  1 : Simulator time (unit s)
#  2 : Left command (unit pwm : 10 bits [0 1023])
#  3 : Right command (unit pwm : 10 bits [0 1023])
#  4 : X coordinate of the robot (unit m)
#  5 : Y coordinate of the robot (unit m)
#  6 : Z coordinate of the robot (unit m)
#  7 : Distance measured by front sonar (unit m)
#  8 : Distance measured by left sonar (unit m)
#  9 : Distance measured by back (rear) sonar (unit m)
# 10 : Distance measured by right sonar (unit m)
# 11 : Number of ticks counted by left odometer (signed integer 32 bits)
# 12 : Number of ticks counted by right odometer (signed integer 32 bits)
# 13 : X component of measured magnetic field (signed integer 32 bits)
# 14 : Y component of measured magnetic field (signed integer 32 bits)
# 15 : Reflectivity of left line sensor (float between 0.0 and 1.0)
# 16 : Reflectivity of middle line sensor (float between 0.0 and 1.0)
# 17 : Reflectivity of right line sensor (float between 0.0 and 1.0)
# 18 : Battery level (float between 0.0 and 1.0)
```

Tutoriel TD2

"log file" : modification de plot_mission.py

- création de "plot_mission_reflectivity_sensors_q2.py" en s'inspirant de "plot_mission.py"
- retrouver l'endroit ou est enregistré "rob1a.log" (dans le dossier "qualif2/A" par ex.)

```
# find on your computer the location of the log file "rob1a.log"
# and set the variable "logfile"
# example on windows :
# logfile = "C:\Users\myusername\Documents\BCA\bca203\scenes\qualif2\A\rob1a.log"
# example on MaxOS :
# logfile = "/home/myusername/Documents/ue22bca/bca203/scenes/qualif2/A/rob1a.log"
# here we are on linux
logfile = "../../../../vrep/define-track-with-remote-api/challenges/qualif2/A/rob1a.log"
# log file can also be set in python execution command line with parameter :
try:
    logfile = sys.argv[1]
except:
    pass
# check if file exists , if not exit ...
if not os.path.exists (logfile):
    print (logfile, "has not been found exit ...")
    exit()
```

Tutoriel TD2

"log file" : modification de plot_mission.py

- sélection des données (numéro de colonne)
- création des tableaux pour récupérer les données

```
# column number (-1) of the data to be analyzed
it = 0  # time
ix = 3 # robot location (x coordinate)
iy = 4 # robot location (y coordinate)
irl = 14 # left reflectivity sensor
irm = 15 # middle reflectivity sensor
irr = 16 # right reflectivity sensor

# create empty arrays to store data
vx=[]
vy=[]
vt=[]
vrl=[]
vrm=[]
vrr=[]
```

Tutoriel TD2

"log file" : modification de plot_mission.py

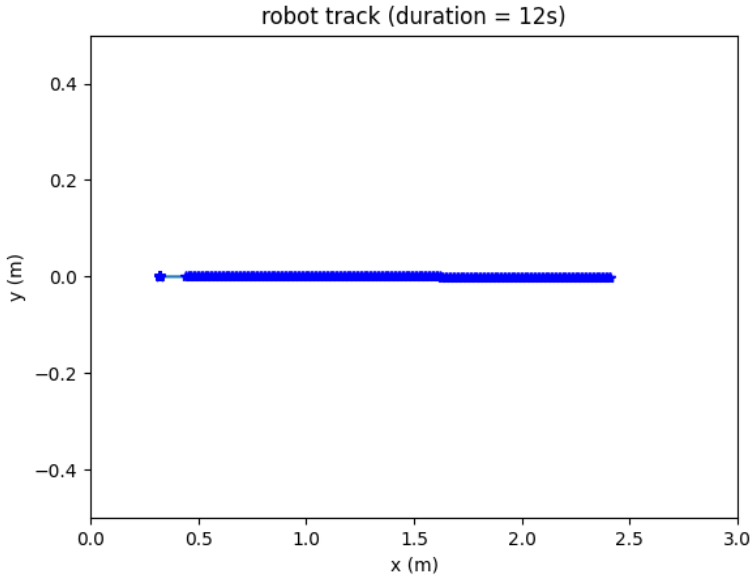
- ouverture du fichier et récupération les données

```
# open the log file
flog = open (logfile,"r")
st = flog.readline()
# read and count (with n) the data
n = 0
while True:
    st = flog.readline() # read a line
    if len(st) == 0:
        break
    v = st[0:-1].split(";") # split the line using ;
    timsim=cvt_float(v[it])
    vt.append(timsim)
    vx.append(cvt_float(v[ix]))
    vy.append(cvt_float(v[iy]))
    vrl.append(cvt_float(v[irl]))
    vrm.append(cvt_float(v[irm]))
    vrr.append(cvt_float(v[irr]))
    n = n+1
flog.close()
print (n,"data points")
```

Tutoriel TD2

"log file" : affichage des données

- affichage de la trajectoire (x,y)



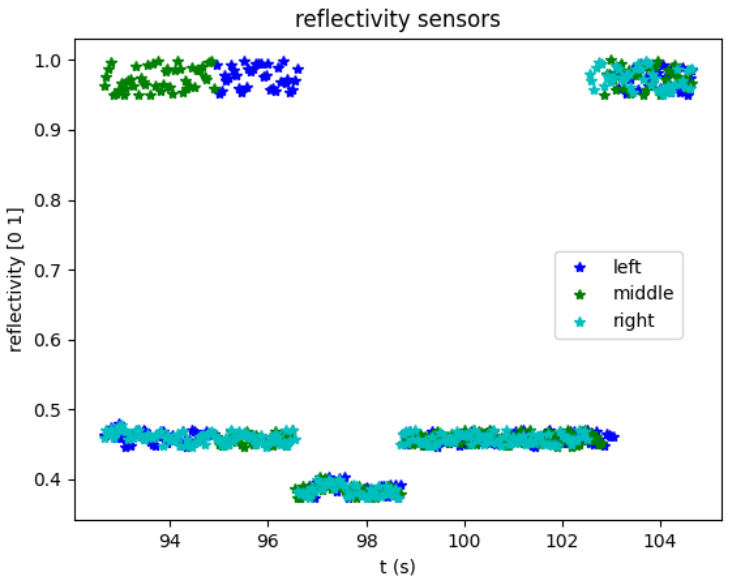

```
# show the track on figure 1
plt.figure(1)
# set axis range :
plt.xlim([-0.5+np.round(np.min(vx)*2.0)/2.0, 0.5+np.round(np.max(vx)*2)/2.0])
plt.ylim([-0.5+np.round(np.min(vy)*2.0)/2.0, 0.5+np.round(np.max(vy)*2)/2.0])
plt.plot(vx,vy) # plot line
plt.plot(vx,vy,'*b') # add markers
plt.xlabel("x (m)")
plt.ylabel("y (m)")
plt.title("robot track (duration = %ds)"%(int(round(vt[-1]-vt[0]))))
plt.savefig('robot_log_track.png')
```

Tutoriel TD2

"log file" : affichage des données

- 3 capteurs de refléctivité
- au début, en ligne droite, capteur central actif
- ensuite, virage à gauche, capteur gauche actif

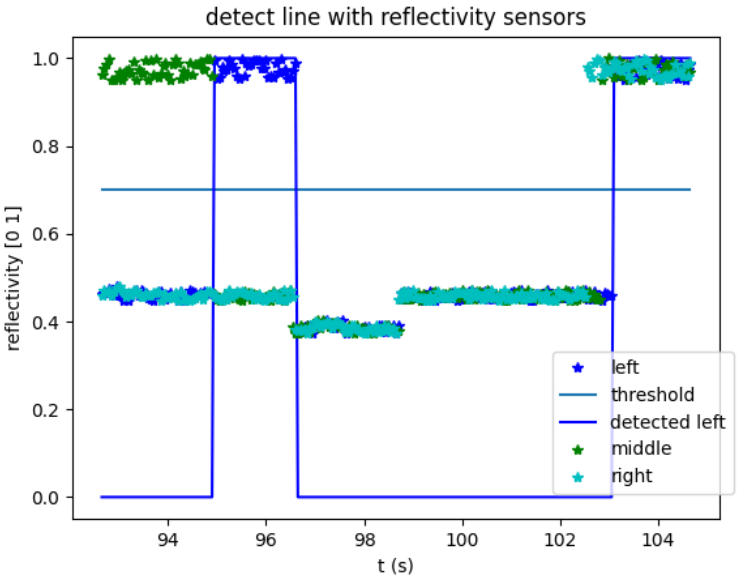
```
# show the proximity sensors on figure 2
fig2 = plt.figure(2)
plt.plot (vt,vrl,'*b',label="left")
plt.plot (vt,vrm,'*g',label="middle")
plt.plot (vt,vrr,'*c',label="right")
fig2.legend(loc=[0.7,0.4])
plt.xlabel("t (s)")
plt.ylabel("reflectivity [0 1]")
plt.title("reflectivity sensors")
```



Tutoriel TD2

"log file" : affichage des données

- définition d'un seuil de 0.7
- valeur > 0.7 -> capteur actif
- capteur gauche actif
- contrôle -> tourner à gauche




```
# process the proximity sensors and show the result
threshold = 0.7 # apply a threshold
fig3 = plt.figure(3)
plt.plot (vt,vrl,'*b',label="left")
plt.plot ([vt[0],vt[-1]],[threshold,threshold],label="threshold")
plt.plot (vt,vrl > threshold,'b',label="detected left")
plt.plot (vt,vrm,'*g',label="middle")
plt.plot (vt,vrr,'*c',label="right")
fig3.legend(loc=[0.7,0.15])
plt.xlabel("t (s)")
plt.ylabel("reflectivity [0 1]")
plt.title("detect line with reflectivity sensors")
```

"plot_mission_reflectivity_sensors_q2.py" est disponible sur MOODLE

Tutoriel TD2

qualif2 : bien débuter la boucle de contrôle pour qualif2A (1/2)

- reprendre la fonction **follow_center_line_v0()** et la modifier en **follow_center_line_v1()**
- modifier les paramètres d'appel de **follow_center_line_v1()**
  - ajouter un paramètre pour changer la direction du robot
  - ajouter un paramètre pour arrêter le robot à une distance donnée du mur frontal
  - facultatif : enlever le paramètre d'arrêt au bout d'une durée maximale de mission, car, en principe, le robot s'arrête face au mur
- lorsque le capteur de réflectivité gauche est actif, le robot doit tourner à gauche.

Tutoriel TD2

qualif2 : bien débuter la boucle de contrôle pour qualif2A (2/2)

- la vitesse du robot est définie par la commandes des moteurs gauche et droite, **spdl** et **spdr**, en utilisant **rb.set_speed(spdl,spdr)**
- la commande des moteurs (spd) peut se répartir en deux composantes :
  - linéaire **spd_lin** (≥ 0) : identique sur les deux roues pour avancer en ligne droite
  - différentielle **spd_dif** : de module identique et de sens opposé sur les deux roues pour changer de direction
  - la commande des moteurs devient : **rb.set_speed(spd_lin+spd_dif,spd_lin-spd_dif)** avec spd_dif < 0 pour tourner à gauche
- les données peuvent être visualisées et analysées en modifiant "plot_mission.py", un exemple est donné avec "plot_mission_first_turn_ok_q2.py".

Tutoriel TD2

Récapitulatif de la mission premier virage

```
import rob1a_v03 as rob1a  # get the robot code
import control  # robot control functions
import filt # sensors filtering functions
import numpy as np
import time

if __name__ == "__main__":
    rb = rob1a.Rob1A()    # create a robot (instance of Rob1A class)
    rb.set_pseudo ("myAwesomePseudo") # you can define your pseudo here (if
    ctrl = control.RobotControl() # create a robot controller

    # log data to analyse sensors
    rb.set_log(1)    # do not log data (set to 1 for logging data)

    # follow the center line until the front wall
    stop_dist = 0.25
    spd_lin = 400
    spd_dif = 0.05*spd_lin
    duration_max = 20.0 # set a max duration of 20 seconds
    ctrl.follow_center_line_v1 (rb,stop_dist,spd_lin,spd_dif,duration_max)

    rb.full_end() # clean end of simulation
```

Tutoriel TD2

Boucle de contrôle

- Dans le module **control.py**, ajouter la boucle de contrôle **follow_center_line_v1()**
- Décrire la fonction pour simplifier son utilisation future
- Code Python

```
def follow_center_line_v1 (self,rb,stop_dist,spd
    """
    Follow the white center line using IR reflec
    Stop at 25 cm of a front obstacle (end wall)

    Input parameters :
    rb : robot
    stop_dist : distance min to front obstacle
    spd_lin : linear speed
    spd_dif : differential speed (to turn)
    duration_max : maximum duration of the move

    Output parameters :
    None
    """
```

Tutoriel TD2

Boucle de contrôle

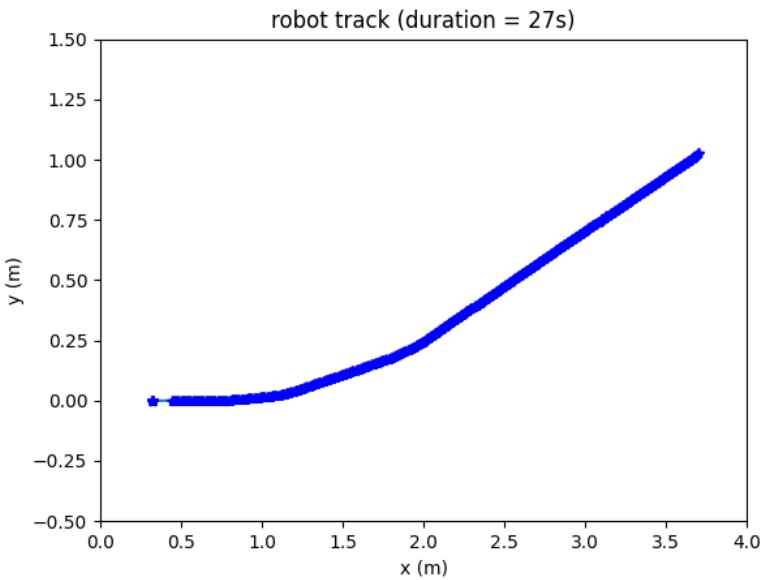
- Si capteur gauche actif (> 0.7), tourner à gauche
- Code Python de la fonction **follow_center_line_v1()**

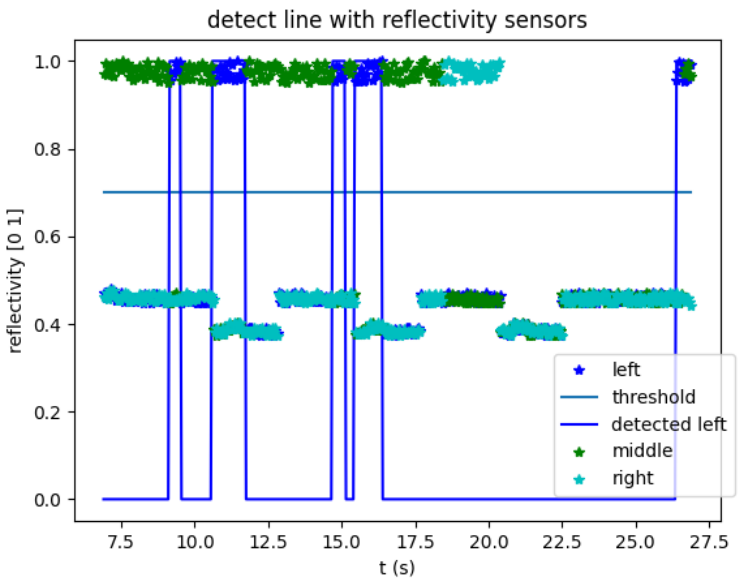
```
loop_iter_time = 0.1 # control at 10 Hz (10 commands/second)
threshold = 0.7 # line detector on if > threshold
t_start = time.time()
while True:
    t0loop = time.time()
    if (time.time() - t_start) > duration_max:
        break # max time reached , escape the loop ...
    # acquire and process sensors
    rl,rm,rr = rb.get_centerline_sensors()
    detect_left = rl > threshold
    d_spd_left = 0
    d_spd_right = 0
    if detect_left:
        d_spd_left = -spd_dif
        d_spd_right = spd_dif
    # set actuators
    rb.set_speed(spd_lin+d_spd_left,spd_lin+d_spd_right)
    # end of loop
    t1loop = time.time()
    dt_sleep = loop_iter_time - (t1loop - t0loop)
    if (dt_sleep > 0):
        time.sleep(dt_sleep) # wait to have perfect loop duration
    else:
        print ("too much computation in this loop, increase loop_iter_time or simplify computation
# stop the robot
rb.stop()
```

Tutoriel TD2

"log file" : affichage des données

- vérification du changement de trajectoire
- virage à gauche OK
- vérification du bon fonctionnement du seuil de 0.7 sur le capteur gauche de réflectivité





Tutoriel TD2

"log file" : affichage des données

- modification de "plot_mission.py" pour ajouter la visualisation des commandes des moteurs

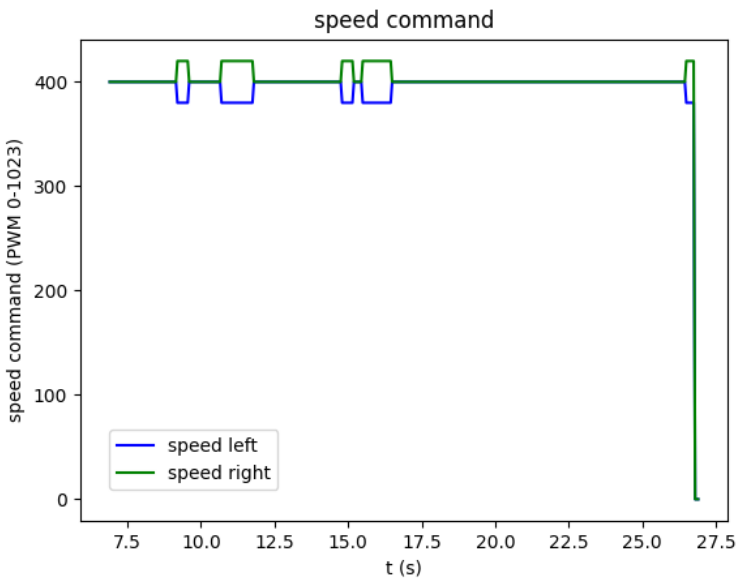
```
# column number (-1) of the data to be analyzed
ispdl = 1 # speed command left
ispdr = 2 # speed command right

# create empty arrays to store data
vspdl=[]
vspdr=[]
# read and count (with n) the data
n = 0
while True:
    st = flog.readline()
    if len(st) == 0:
        break
    v = st[0:-1].split(";")
    timsim=cvt_float(v[it])
    vt.append(timsim)
    vx.append(cvt_float(v[ix]))
    vy.append(cvt_float(v[iy]))
    vrl.append(cvt_float(v[irl]))
    vrm.append(cvt_float(v[irm]))
    vrr.append(cvt_float(v[irr]))
    vspdl.append(cvt_float(v[ispdl]))
    vspdr.append(cvt_float(v[ispdr]))
    n = n+1
flog.close()
```

Tutoriel TD2

"log file" : affichage des données

- vérification de la commande des moteurs
- commande différentielle lorsque que le capteur gauche est actif
- "plot_mission_first_turn_ok_q2.py" est disponible sur MOODLE

```
# show the speed command to check the control l
fig5 = plt.figure(5)
plt.plot (vt,vspdl,'b',label="speed left")
plt.plot (vt,vspdr,'g',label="speed right")
fig5.legend(loc=[0.16,0.16])
plt.xlabel("t (s)")
plt.ylabel("speed command (PWM 0-1023)")
plt.title("speed command")
plt.savefig('robot_log_speed_cmd.png')
```



Tutoriel TD2

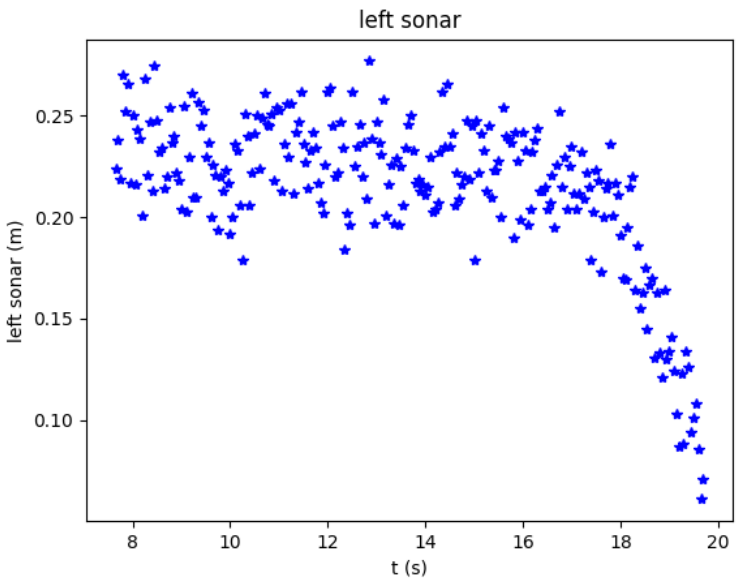
Bien débuter le challenge qualif3

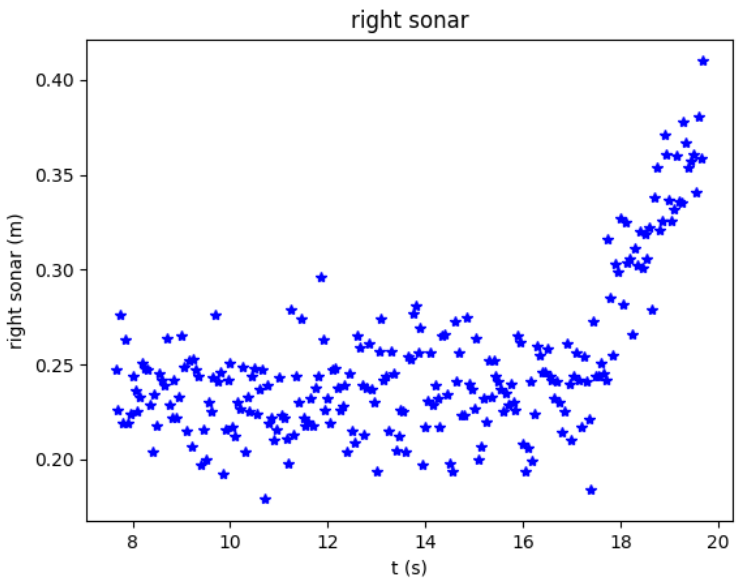
- Approche simple en plusieurs boucles de contrôle
- Exemple avec 4 boucles de contrôle :
  - **walls_follow** : follow the 2 walls to stay at the center of the track
  - **distance_to_front_wall** : set the robot at a given distance to the front wall
  - **inplace_turn** : search for the free side (no wall) and turn (in place) of 90 degrees to set the robot in the free direction
  - **straight_until_walls** : go straight until detection of the 2 walls (to use **walls_follow**)
- Exécuter une séquence de ces 4 boucles de contrôle pour réaliser **qualif3**
- Définir et analyser les "bons" capteurs pour chaque boucle de contrôle
- Utiliser la télémétrie
- Mise au point en déplaçant le point de départ du robot

Tutoriel TD2

Suivre les deux murs : **walls_follow**

- exemple : utiliser les sonars (gauche et droit) pour conserver la distance aux murs
- test : avancer tout droit pendant 12 secondes et utiliser la télémétrie pour visualiser les données
- la distance mesurée par sonar gauche décroît, et celle du sonar droit augmente
- on peut utiliser ces variations pour corriger la direction du robot
- si les données sont trop bruitées, il faudra les filter

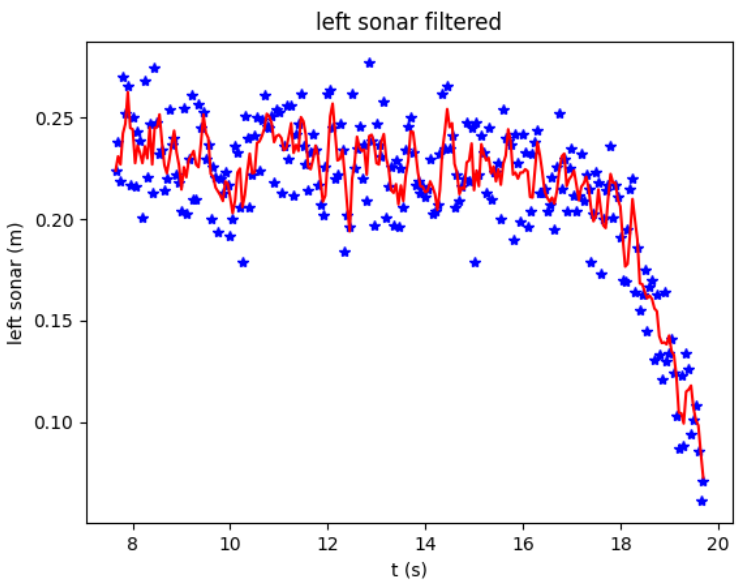


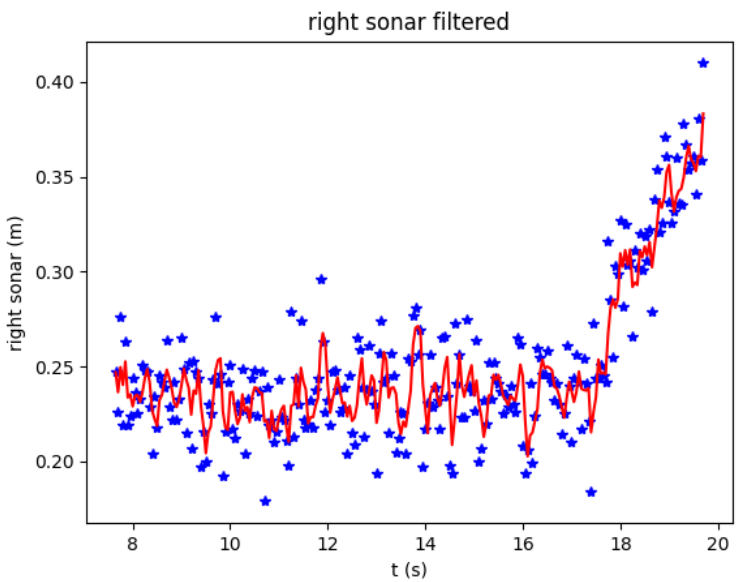


Tutoriel TD2

Suivre les deux murs : **walls_follow**

- exemple de filtre pase bas très simple : "filt_tuto_lab2_q3.py" (MOODLE)
- utiliser la télémétrie pour tester le filtre : "plot_mission_sonars_q3.py" (MOODLE)
- le filtre (en rouge) améliore la mesure, sera-ce suffisant ?
- si non concluant, programmer les filtres décrits dans le sujet de TD





Tutoriel TD2

Tourner sur place de 90 degrés : **inplace_turn**

- exemple : utiliser les sonars (gauche et droit) pour savoir de quel coté tourner
- utiliser les odomètres pour tourner
- éventuellement : vérifier l'absence de bruit sur les odomètres
- autre solution : utiliser la boussole
- problème : il faut "calibrer" la boussole avant de l'utiliser ...

Tutoriel TD2

Changer ke point de départ du robot

- afin de mettre au point les différentes boucles de contrôle, il peut être utile de déplacer le point de départ du robot
- le point de départ est défini pas les coordonnées x et y
- l'orientation est définie par rapport à l'axe z (vertical)
- vidéo d'explication : https://moodle.ensta-bretagne.fr/mod/resource/view.php?id=60287
- attention : effet indésirable, la position du robot (x,y) dans le "log file" ne fonctionne plus après le déplacement du point de départ