



MODULE 1 - CONTRÔLE BAS NIVEAU ROBOT MOBILE

SPE/ROB - UE32 - 2020-2021

Le robot DART - historique

Remplacement en 2016 du robot JOG en service depuis 2010 ...

... par le robot DART

Robot prototype réalisé par les étudiants de CI2016 dans le cadre des UV projets de 2A 3.4 et 4.4 :

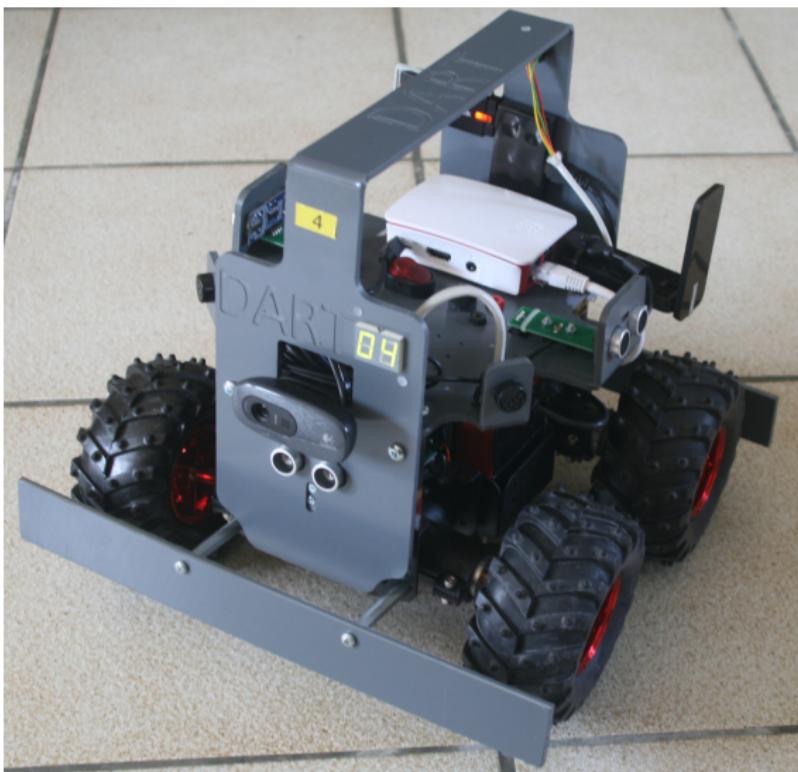
- D : Brian **Detourbet**
- A : Fahad **Al Shaik**
- A : Elouan **Autret**
- R : Corentin **Rifflart**
- R : Clément **Rodde**
- T : Rémi **Terrien**

DAARRT simplifié en DART

2011-2015 JOG, 2016-2017 DART (V1)



2018 - Evolution - DART V2



2018 - Evolution DART V2

Retour d'expérience DART :

- Robot très nerveux → casse mécanique (réducteurs)
- Carte CPU PCDUINO3 surchargée (acquisition capteurs ultrasons & vidéo)

Modifications DART V2 :

- sécurité : carrosserie, poignée , passage de 12V à 6 V
- électronique embarquée : ajout Raspberry PI 3 (traitements audio et vidéo), cartes micro-contrôleur (acquisitions ultrasons et encodeurs, afficheur 7 segments)

DART V2b : robot pédagogique de base de la spécialité Robotique (2A à partir de la rentrée 2019)

Le robot DART V2

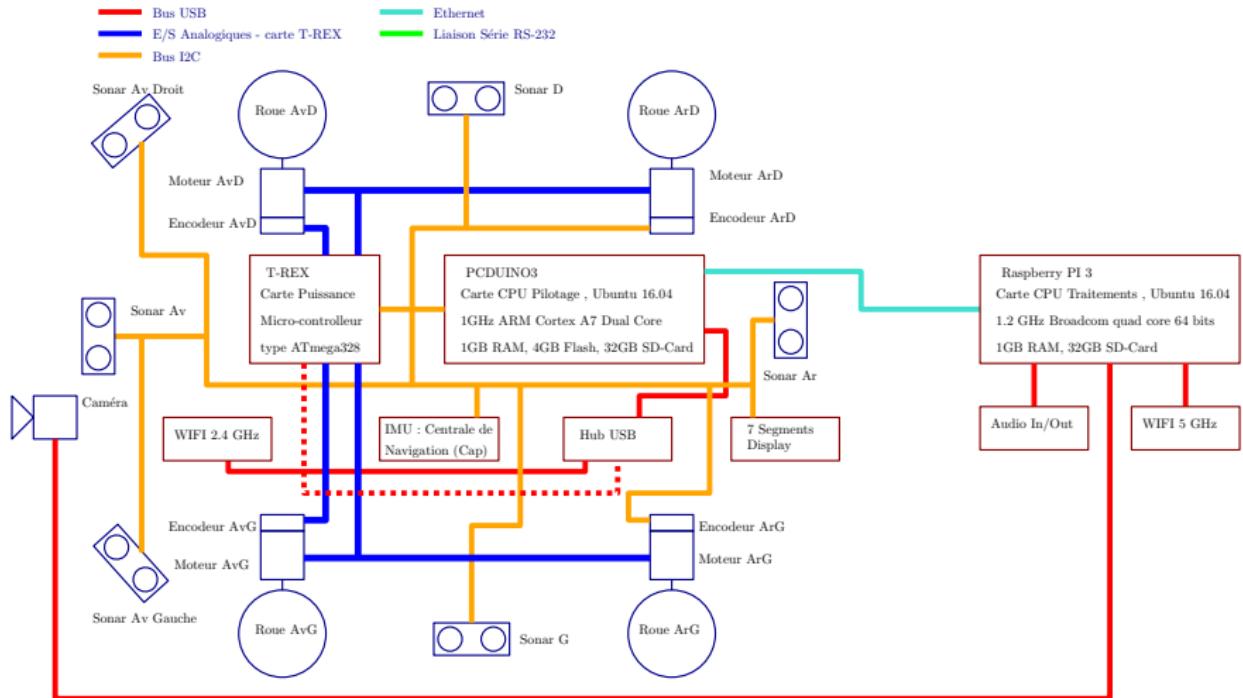
Le contrôle du robot DART V2 se fait par l'acquisition des informations issues des capteurs (sonars,odomètres,centrale de navigation inertie, son et vidéo) et la commande des actionneurs (moteurs, afficheur 7 segments).

Plusieurs méthodes d'accès "haut niveau" aux capteurs et actionneurs cohabitent sur le robot DART V2 :

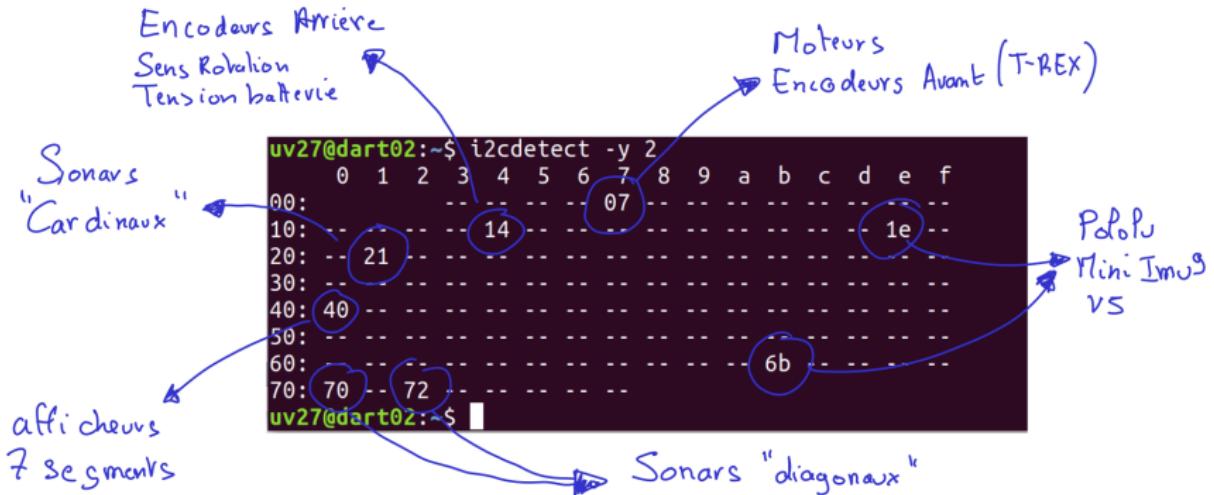
- bus I2C
- liaisons séries USB

L'accès "bas niveau" se fait à travers plusieurs cartes micro-contrôleur (PIC et ARDUINO like) programmables

Robot DART V2 - Architecture



Robot DART V2 - Bus I2C



Contrôle du robot DART V2

Les fonctions de bas niveau d'accès aux pilotes, à la mémoire et aux bus sont fournies. Il faudra écrire les fonctions de haut niveau, par exemple, pour la centrale de navigation (IMU : Inertial Motion Unit) connectée au bus I2C :

- nous disposons de la liste des commandes à envoyer,
- nous disposons du format des réponses (accéléros, gyros, cap mag.),
- nous disposons du module smbus (I2C) de Python3 pour accéder au capteur par le bus I2C,
- nous devons définir la configuration du capteur (setup) en écrivant les bonnes valeurs dans les registres,
- nous devons écrire, par exemple, une fonction lectureCompas() en envoyant la bonne commande sur le bus, puis récupérer les informations et les mettre au format qui nous intéresse,
- éventuellement, nous devrons étalonner (calibrer) et filtrer les informations délivrées par le capteur.

En fin de cette partie de TD : vous disposerez d'un ensemble de fonctions Python3 vous permettant d'acquérir de façon fiable les informations capteurs (boussole, accéléromètres, gyroscopes, encodeurs, tension batterie) et de piloter les actionneurs du robot DART (moteurs gauches et droits, afficheur 7 segments).

Simulateur - Mise au point

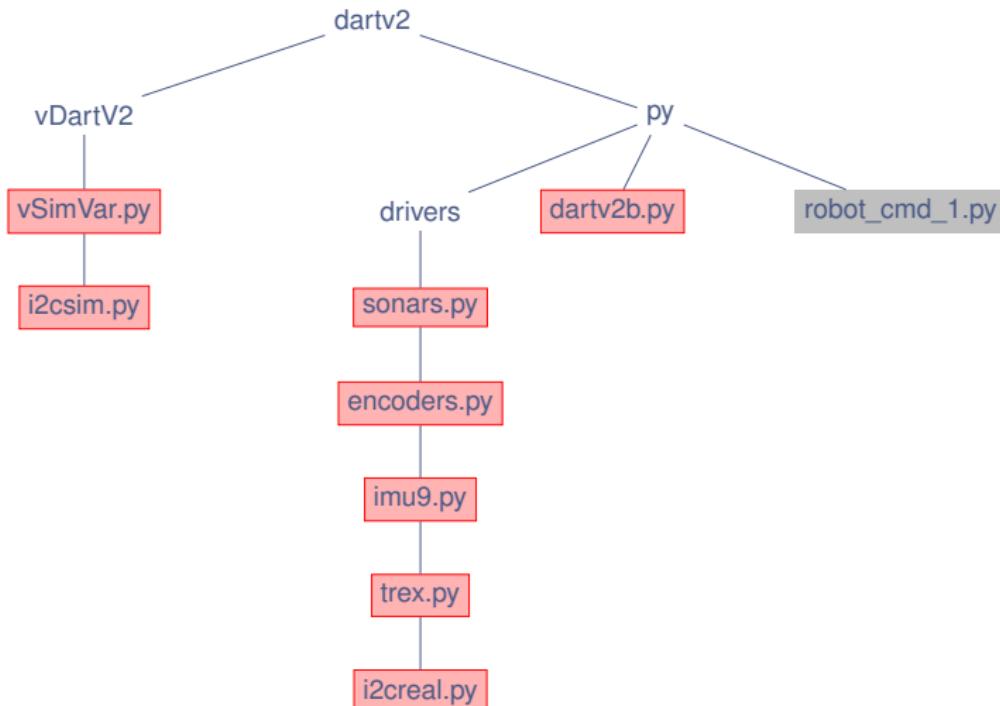
- Nous utiliserons un simulateur programmé à l'aide de V-REP. Le simulateur sera lancé dans un terminal séparé.
- Le simulateur est écrit en langage LUA dans V-REP dont la version électronique est téléchargeable gratuitement depuis :
<http://coppeliarobotics.com/downloads.html>



- En principe vous utiliserez le même programme sur le simulateur et sur le robot réel. Le programme détecte si il est exécuté sur le robot, sinon il se connecte au simulateur.
- Programmation en Python 3
- Le simulateur permet (en principe !) de commander les moteurs et fournit les informations issues des capteurs avec les mêmes fonctions que le robot réel.

Architecture Logicielle DART V2

Sur le robot, seul le dossier py est utilisé.



DART V2 - Connection et exécution

- Se connecter sur le robot (YY est le numéro du robot et le mot de passe est uv27) :

```
ssh uv27@172.20.25.1YY
```

- Créer votre dossier sur le robot (XXXX est le pseudo de votre groupe),

```
mkdir grp_XXXX
```

- Quitter le robot en tapant ctrl-D

- Se placer dans le dossier tests sur votre PC

- Copier les programmes sur le robot :

```
scp -r * uv27@172.20.25.1YY :grp_XXXX
```

- Se reconnecter sur le robot et exécuter le programme (ex. dart_bais_fun.py) :

```
ssh uv27@172.20.25.1YY
```

```
cd grp_XXXX
```

```
python3 dart_basic_fun.py
```

Contrôle-Commande

Contrôle-commande du robot DART :

- Définition d'une mission
- Formalisme de la machine à états finis (FSM : Finite State Machine)
- Pilotage du DART par FSM (module 2)

Exécution de la mission : démarrage à l'état START et fin à l'état END et exécutant tous les Δt ms :

- Estimation des erreurs entre le comportement souhaité et le comportement réellement exécuté
- Définition des corrections
- Commande des actionneurs (moteurs) pendant Δt
- Acquisition des informations capteurs
- Définition de l'évènement suivant

Gestion du temps

- Importer le module time de Python :

```
import time
```

- Acquérir le temps (donné par l'horloge de la carte)

```
t0 = time.time()      # value in seconds (float)
st0 = time.asctime()  # string of chars
```

- Dormir pendant un temps de t1 secondes (float)

```
time.sleep(t1)
```

- Exécuter une action qui dure **delta_t** secondes :

```
import time
delta_t=0.05
t0=time.time()
do_my_action # work to be done
while (time.time()-t0) < delta_t:
    pass # or time.sleep(0.0001)
print (time.time()-t0)
```

Méthodes de Régulation

Appliquer une correction en fonction de l'erreur entre :

- le comportement souhaité : **la consigne**
- le comportement réel : **la mesure**

Mode "Bang-Bang" :

- Utilisation du signe de l'erreur pour la correction
- Module de la correction constant

Mode Proportionnel :

- Utilisation du signe de l'erreur pour la correction
- Module de la correction est une proportion de l'erreur
- Coefficient de proportionnalité K_P

Régulateur Proportionnel

Soit **headOk** le cap désiré exprimé dans un repère géographique (0° = Nord)

Mesure du cap à l'aide de la boussole avec une résolution de 0.1°

headActual=compass.getWordHeading()

Calcul de l'erreur de cap :

headErr = deltaCap (headOk,headActual)

Commande des moteurs :

speedLeft = **speedLinear** + **Kp*****HeadErr**

speedRight = **speedLinear** - **Kp*****HeadErr**

Détermination empirique de **Kp**, si **Kp** faible le robot met plus de temps pour atteindre le bon cap, si **Kp** élevé le robot risque de d'osciller autour du bon cap ...

Attention, le résultat du calcul de la différence de cap, fonction deltaCap(), doit être compris entre -180 et $+180$ degrés

Remarque : si **speedLinear** = 0, le robot tourne sur lui-même, si **speedLinear** est non nul le robot régule son cap en avançant ...

Régulateur PID

La correction est la somme de trois termes :

- un terme (pondéré par K_P) proportionnel à l'erreur
- un terme (pondéré par K_D) proportionnel à la dérivée de l'erreur (variation de l'erreur)
- un terme (pondéré par K_I) proportionnel à l'intégrale de l'erreur (dérive de l'erreur)

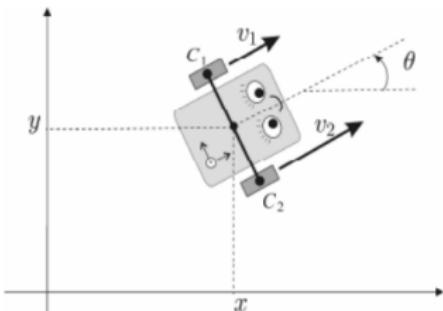
Définition expérimentale des gains K_P , K_D et K_I

```
previous_error = setpoint - actual_position
integral = 0
start:
    error = setpoint - actual_position
    integral = integral + (error*dt)
    derivative = (error - previous_error)/dt
    output = (Kp*error) + (Ki*integral) + (Kd*derivative)
    previous_error = error
    wait(dt)
    goto start
```

Rappel Modèle Char

JOG : modèle de type char

$$\begin{cases} \dot{x} = v \cos \theta \\ \dot{y} = v \sin \theta \\ \dot{\theta} = w \end{cases}$$



r Rayon des roues

l Distance entre roues

Modèle un peu trop simple pour le DART V2 car ne tient pas compte du "drift"

Rappel Modèle Char

$$v = \frac{v_1 + v_2}{2}$$

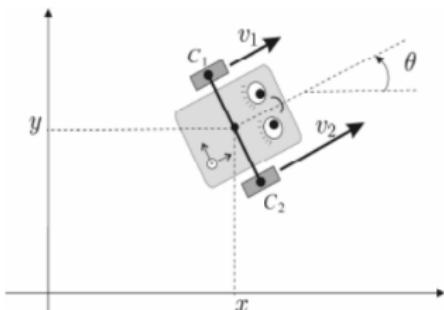
$$\omega = \frac{v_2 - v_1}{l}$$

$$\omega_1 = \alpha_1 u_1$$

$$\omega_2 = \alpha_2 u_2$$

$$v_1 = r\omega_1$$

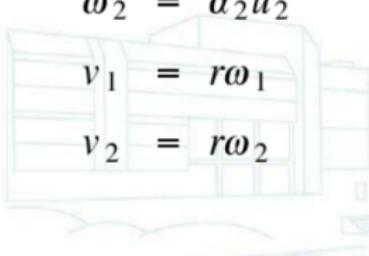
$$v_2 = r\omega_2$$



$$\begin{cases} \dot{x} = v \cos \theta \\ \dot{y} = v \sin \theta \\ \dot{\theta} = w \end{cases}$$

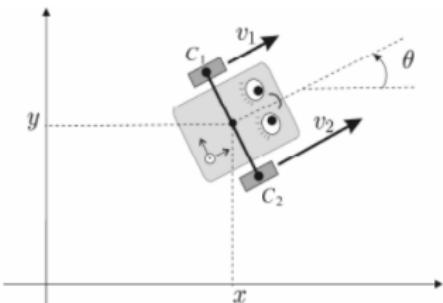
r Rayon des roues

l Distance entre roues



Rappel Modèle Char

$$\begin{cases} \dot{x} = \frac{r\alpha_1 u_1 + r\alpha_2 u_2}{2} \cos \theta \\ \dot{y} = \frac{r\alpha_1 u_1 + r\alpha_2 u_2}{2} \sin \theta \\ \dot{\theta} = \frac{r\alpha_2 u_2 - r\alpha_1 u_1}{l} \end{cases}$$



$$v = \frac{v_1 + v_2}{2}$$

$$\omega = \frac{v_2 - v_1}{l}$$

$$\omega_1 = \alpha_1 u_1$$

$$\omega_2 = \alpha_2 u_2$$

$$v_1 = r\omega_1$$

$$v_2 = r\omega_2$$

$$\begin{cases} \dot{x} = v \cos \theta \\ \dot{y} = v \sin \theta \\ \dot{\theta} = w \end{cases}$$

r Rayon des roues

l Distance entre roues

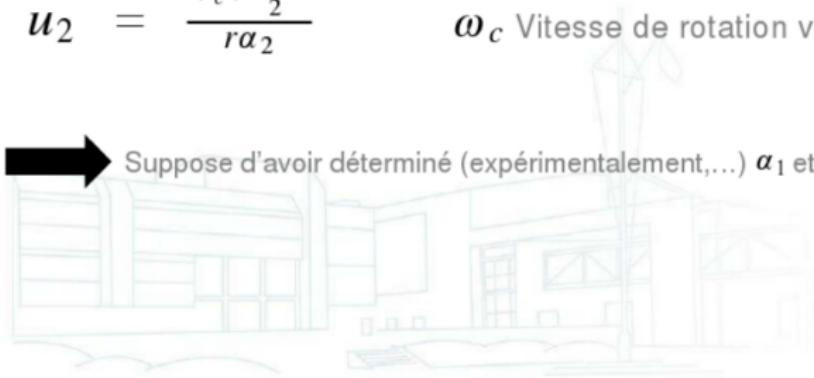
Vitesse Tangentielle

$$u_1 = \frac{v_c - \frac{\omega cl}{2}}{r\alpha_1} \quad v_c \text{ Vitesse tangentielle voulue}$$

$$u_2 = \frac{v_c + \frac{\omega cl}{2}}{r\alpha_2} \quad \omega_c \text{ Vitesse de rotation voulue}$$



Suppose d'avoir déterminé (expérimentalement,...) α_1 et α_2



Trajectoire Circulaire

On a la relation: $v_c = R\omega_c$

v_c Vitesse tangentielle voulue

R Rayon du cercle à suivre

En remplaçant dans la formule de la commande par vitesse tangentielle et de rotation:

$$u_1 = \left(1 - \frac{l}{2R}\right) \frac{v_c}{r\alpha_1}$$

$$u_2 = \left(1 + \frac{l}{2R}\right) \frac{v_c}{r\alpha_2}$$

Trajectoire Linéaire

Vu que $v = \frac{d}{t}$

En imposant d et t cela revient à une commande en vitesse tangentielle (avec vitesse de rotation nulle pour aller tout droit)



Odomètres - Position Angulaire

$$\theta_1 = \beta_1 odo_1$$

$$\theta_2 = \beta_2 odo_2$$

Avec β_1 et β_2 à déterminer (expérimentalement,...)



Odomètres - Vitesse Angulaire

$$\omega_{m1} = \frac{\beta_1 \delta odo_1}{dt}$$

$$\omega_{m2} = \frac{\beta_2 \delta odo_2}{dt}$$

Avec β_1 et β_2 à déterminer (expérimentalement,...)



Odomètres - Régulation

Boucle faisant:

$$u_1 = K_1(\omega_{c1} - \omega_{m1})$$

$$u_2 = K_2(\omega_{c2} - \omega_{m2})$$

→ Commande grand gain (mais risque d'être saccadée)

→ Ceci n'est pas vraiment une régulation du char, mais plutôt de ses moteurs

Suppose qu'on a β_1 et β_2 mais pas α_1 et α_2