UABot v1.0
Guia ràpida d'usuari

Joan Oliver Malagelada,
Ricardo Toledo Morales,
Josep Maria Sanz Xicola

Curs 2010

Índex

1 Introducció UABot

UABot és una plataforma robòtica de petites prestacions que incorpora elements sensorials i motrius. L'establiment d'un enllaç de comunicació amb una estació de treball mestre ens permet interactuar amb la plataforma i el seu entorn.

Basat en el microcontrolador ATMega128 d'Atmel, l'UABot incorpora en la memòria Flash un sistema operatiu en temps real anomenat FreeRTOS. És un projecte open source amb llicència GPL i suport múltiplataforma.
Junt amb el sistema operatiu, l'acompanya una API que ens permet crear múltiples tasques per execució concurrent, sincronització, mecanismes d'exclusió mútua, creació de cues amb prioritats, etc...

Actualment, la nostra plataforma UABot executa tres tasques:

- Tasca Comunicació (prioritat alta)
- Tasca Motors (prioritat mitjana)
- Tasca Sensors (prioritat baixa)

Cada tasca gestiona una parcel·la de control i actuació del microcontrolador.

El control del robot es fa a través d'una estació de treball, que és l'encarregada de transmetre les accions desitjades per part de l'usuari. Per aconseguir un mecanisme de control i consulta senzills s'ha desenvolupat en Python un conjunt de classes que permeten el govern de la plataforma de forma clara i simple.

1.1 Instal·lació i posada en marxa

Descomprimir l'arxiu `pyuabot.zip` en una carpeta (p.e.: `C:\\mybot`). A continuació, en la consola del Python, carregar el `path` on hem descomprimit `pyuabot.zip`:

```
>>> import sys
>>> sys.path.append("C:\\mybot")
>>> sys.path  # Comprovem que el path s'ha inclós
```

Ara podem importar el mòdul `uabot`,

```
>>> import uabot
```

`uabot` importa tota una sèrie de mòduls que ens proporcionen la funcionalitat i les eines necessaries per consultar i actuar amb el robot. Principalment dispondrem de mètodes de control de motors i mètodes de consulta de sensors.

1.2 Inici

Creem una instància d'un robot.

```
uabot.Uabot(name="UABOT")
```

La classe `uabot.Uabot(...)` genera la instància d'un robot. Opcionalment podem assignar un nom representatiu (per defecte és **UABOT**). Exemple:

```
>>> ua = uabot.Uabot("MyBot")
>>> ub = uabot.Uabot()
>>> print ua.name
MyBot

>>> print ub.name
UABOT
```

`ua` és l'instància d'un robot amb nom **Mybot**. `ub` és l'instància d'un altre robot amb el nom per defecte **UABOT**.
Establim la comunicació `ua` - robot utilitzant el port `COM4`.

```
>>> ua.connect("COM4")
```

Si tot va bé, ha d'apareixer per pantalla el següent missatge:

```
M-Task: OK
S-Task: OK
```

Arribats a aquest punt, tenim la instància `ua` vinculada al nostre robot físic, podent enviar i rebre dades del robot.

1.3 Puntualitzacions

Tots els mètodes propis de la plataforma UABot tenen per conveni retornar un valor. Si per algun motiu la crida del mètode no reeixís, el mateix mètode invocat llençaria una excepció de la naturalesa escaient.
És importat distingir la diferència entre una devolució `False` i la generació d'una excepció. En el primer cas, totes les accions pertinents al mètode han

pogut ser resoltes i la resposta per part del robot ha estat un opcode d'error (comanda desconeguda, robot ocupat...). En el segon cas, l'execució del mètode no ha pogut finalitzar correctament, possiblement quedant l'acció a mig fer. Amb un retorn `False` el robot és troba operatiu i simplement ens indica que l'acció no ha pogut ser. Una excepció és producte d'un incident sever on possiblement el robot no respongui a les nostres peticions.

Sempre que parlem de posicions i direccions (esquerra, dreta, davant i enrere) la referència és presa segons el punt de vista del robot, no segons el nostre punt de vista. Per tant, quan parlem d'esquerra, ens referim a l'esquerra del robot (la nostra dreta si el contemplem frontalment). Una manera fàcil de pensar-ho és imaginar que nosaltres anem muntats sobre el robot, llavors els dos punts de referència coincideixen.

2 Treball amb motors

Els motors venen caracteriztzats per una objecte membre de `ua` anomentat `motor` que ens permet configurar ambdós motors,

```
>>> ua.motor
(motor.Motor instance at 0xb78828ec)

>>> ua.motor.status
[0, 0]
```

on `[X,Y]`, `X` fa referència a l'estat de la roda esquerra i `Y` referent a l'estat de la roda dreta. Els valors de `X,Y` $\in [-1, 1]$ són valors reals on 0 és velocitat zero, 1 és màxima velocitat de gir en sentit horari i -1 màxima velocitat de gir en sentit antihorari.

Podem consultar quantes comandes enviades/rebudes relacionades amb els motors hem cursat,

```
>>> ua.motor.nReq()
0

>>> ua.motor.nAck()
0
```

2.1 `Motor.forward(vel=1, t=-1)`

El mètode `forward` mou cap endavant amb velocitat `vel` i durada `t` segons. Ambdós arguments són opcionals.
L'argument `vel` permet ajustar la velocitat de gir. Un valor negatiu de `vel` fa el moviment invers —recular—.
`vel` $\in [-1, 1]$. Si `vel` < -1 o `vel` > 1, el comportament és com si fos -1 o 1 respectivament.
`t` permet indicar la durada, en segons, del moviment. Un cop transcorregut `t`, els motors tornen a l'estat anterior a l'invocació del mètode. Si `t` < 0 llavors l'acció té una durada indefinida.
El valor de retorn és `True` o `False` en funció de si l'acció s'ha produït. Exemples:

```
# Avança amb velocitat 0.2 durant 2.34 segons
>>> ua.motor.forward(0.2, 2.34)
True

# Avança amb velocitat 1 durant 0.45 segons
```

```
>>> ua.motor.forward(1.67, 0.45)
True

# Retrocedeix amb velocitat 0.7 durant 0.93 segons
>>> ua.motor.forward(-0.7, 0.93)
True

# Avança amb velocitat 0.5 indefinidament
>>> ua.motor.forward(vel=0.5)
True

# Aturat durant 2 segons i després recupera l'estat
# anterior (vel=0.5)
>>> ua.motor.forward(vel=0, t=2)
True

# Avança amb velocitat 0.25 indefinidament
>>> ua.motor.forward(0.25, -3)
True

# Avança amb velocitat 1 indefinidament
>>> ua.motor.forward()
True
```

2.2 `Motor.backard(vel=1, t=-1)`

Funció inversa a l'anterior.

2.3 `Motor.rotate(vel=1, t=-1)`

Moviment de rotació amb velocitat `vel` i durada `t` segons.
`vel` > 0 gir, en sentit horari (cap a la dreta). `vel` < 0, gir en sentit antihorari (cap a l'esquerra).
Si `vel` $\in [-1, 1]$, fora d'aquest rang, el comportament és com si fos -1 o 1 en funció de si del signe de `vel`.
Si `t` < 0, la rotació amb `vel` tindrà una durada indefinida.

2.4 `Motor.pvl(vel=1, t=-1)`

Pivotació, roda esquerra inmòbil, amb velocitat `vel` durant un temps de `t` segons.

2.5 `Motor.pvr(vel=1, t=-1)`

Pivotació, roda dreta inmòbil, amb velocitat `vel` durant un temps de `t` segons.

2.6 `Motor.move(lwheel, rwheel, t=-1)`

Permet governar la velocitat de gir de les dues rodes de forma independent durant un cert temps `t`.
`lwheel` $\in [-1, 1]$ correspon a la velocitat desitjada de la roda esquerra i `rwheel` $\in [-1, 1]$ a la respectiva roda dreta. Com ja hem vist, valors fora de $[-1, 1]$ tenen un comportament com si fossin -1 o 1.
`t` indica la durada d'aquesta nova configuració. En acabat, recupera l'estat dels motors d'abans de la crida al mètode. Com els anteriors mètodes, un `t` < 0 suposa una durada indefinida del moviment fins a rebre nova ordre.

2.7 `Motor.saveCalibration()`

Per calibrar els motors hem de jugar amb el mètode `move` tot trobant uns valors adequats que equilibrin el control dels motors.
Trobats aquests valors, podem cridar el mètode `saveCalibration` per salvar-los en l'EEPROM del microcontrolador i fer persistent la calibració.
Després de `saveCalibration()`, `status = [0.0, 0.0]`. És convenient que un cop feta la calibració, provocar un hardware-reset per comprovar que els valor de calibració s'han enmagatzemat correctament i tenen l'efecte desitjat.

3 Sensors

El nostre robot conté una sèrie de sensors els quals es poden consultar a través de l'objecte membre `sensor` de `ua`.
Igual que en el cas dels motors, podem consultar quantes comandes hem efectuat:

```
>>> ua.sensor.nReq()
0

>>> ua.sensor.nAck()
0
```

La consulta a un dels sensors disponibles es pot fer a través del mètode `get` o utilitzant `[]` per refèrenciar l'operació desitjada. Exemple:

```
>>> ua.sensor.get("LDR")
(56, 59)

>>> ua.sensor["LDR"] # Equivalent a l'anterior
(56, 59)
```

3.1 `Sensor.listSensorOp()`

Proporciona una llista amb els diferents opcodes de consulta relacionats amb els sensors. Exemple:

```
>>> ua.sensor.listSensorOp()
['LDR', 'IRL', 'IRF', 'LFW', 'MIC']
```

3.2 Sensors disponibles

Observem que actualment disposem de cinc operacions de consulta de sensors[1] diferents:

- `LDR`: Petició de consulta als sensors de llum (**L**ight **D**ependent **R**esistors). El valor de devolució, si la comanda ha esta tractada correctament, és una tupla de dos elements `(X,Y)` on `X` és el valor del LDR esquerra i `Y` el valor del LDR dret.
  Els valors varien en funció de la lluminositat de l'entorn en un rang de 256 valors possibles on, 0 és molta llum i 255 és foscor absoluta.
  `False` quan la comanda enviada ha estat rebutjada pel robot.

[1] Tots els valors retornats d'una consulta de sensors són valors enters

- `IRL`: Consulta els infrarojos laterals. El resultat de la devolució del mètode és una tupla amb dos elements tals com `(X,Y)` on, `X` referent a l'estat de l'infraroig esquerra i `Y` referent a l'estat de l'infraroig dret. `X,Y` $\in [0, 255]$ on 0 és no obstacle i 255 és màxima proximitat.
  `False` quan la comanda enviada ha estat rebutjada pel robot.

- `IRF`: Consulta els infrarojos frontals. Un devolució exitosa de la consulta és una tupla `(X,Y)` on `X` correspon a l'estat de l'infraroig frontal esquerra i `Y` correspon a l'estat de l'infraroig frontal dret. `X,Y` $\in [100, 255]$ on 255 és no obstacle i 100 és màxima proximitat.
  `False` quan la comanda enviada ha estat rebutjada pel robot.
  Nota: L'infraroig frontal pot proporcionar valors incorrectes inferiors a 100. És convenient descartar totes aquelles mostres per sota de 100.

- `LFW`: Consulta l'estat del seguidor de línia. El retorn correcte és una tupla d'un element `(X)`. `X` $\in [0, 15]$. El seguidor de línia esta format per 4 parelles infrarojos emissor/receptor associats a un bit. Així doncs, tindrem la següent configuració de bits: $B_3B_2B_1B_0$ on B_3 sensor esquerra, B_2 sensor centre-esquerra, B_1 sensor centre-dret i B_0 sensor dret del seguidor.
  `False` quan la comanda enviada ha estat rebutjada pel robot.

- `MIC`: Obté una mostra de só. El retorn correcte és una tupla d'un element tal com `(X)`. `X` $\in [0, 255]$.
  Nota: L'aplicació útil d'aquest micròfon no és la de mostrejar audio, és detectar, de forma puntual, un canvi acústic en l'ambient. Normalment, el micròfon proporciona una valor centrat en el `(32)`. Aplicant una certa tol·lerància `tol`, podem determinar que les mostres rebudes fora del valor centrat en `32` $\pm$ *tol* són pertorbacions.

```
# Consulta als LDR
>>> ua.sensor["LDR"]
(75, 73)

# Fem 8 consultes consecutives als IR Laterals
>>> for i in xrange(8): ua.sensor["IRL"]
(30, 23)
(30, 23)
(32, 24)
(31, 23)
(31, 23)
(30, 21)
(31, 22)
(31, 23)
```

```
# Fem 4 consultes consecutives als IR Frontals
>>> for i in xrange(4): ua.sensor.get("IRF")
(255, 255)
(255, 255)
(255, 255)
(255, 255)

# Consulta l'estat del seguidor de línia
>>> ua.sensor.get("LFW")
(15)
```

3.3 `Sensor.setStepMem(k, n)`

La classe `Sensor` té un variable membre anomenada `status`. `status` és un diccionari que emmagatzema les n últimes mostres de cada sensor a mode d'historial.
Les mostres estan ordenades per ordre d'antiguitat. La més recent en primera posició. així successivament.
Quan iniciem per primera vegada, abans d'haver fet cap consulta, `status` ha d'estar buit,

```
>>> ua.sensor.status
{'LDR':[], 'LFW':[], 'IRL':[], 'IRF':[], 'MIC':[] }
```

Després de varies consultes, per defecte status contindrà les 4 últimes mostres de cada sensor. Exemple:

```
>>> ua.sensor.status
{'LDR':[], 'LFW':[], 'IRL':[], 'IRF':[], 'MIC':[] }

# Consultem 3 vegades els LDR
>>> for i in xrange(3): ua.sensor["LDR"]
(75, 73)
(74, 73)
(73, 74)

# Visualitzem les últimes mostres del LDR
>>> ua.sensor.status["LDR"]
[(73, 74), (74, 73), (75, 73)]

# Totes les últimes mostres de tots els sensors
>>> ua.sensor.status
```

```
{'LDR':[[73, 74], [74, 73], [75, 73]], 'LFW':[],
 'IRL':[], 'IRF':[], 'MIC':[] }


# Consultem 6 vegades els IRL
>>> for i in xrange(6): ua.sensor["IRL"]
(30, 23)
(30, 23)
(32, 24)
(31, 23)
(31, 23)
(30, 21)

>>> ua.sensor.status["IRL"]
[(30, 21), (31, 23), (31, 23), (32, 24)]

>>> ua.sensor.status
{'LDR':[(73, 74), (74, 73), (75, 73)], 'LFW':[],
 'IRL':[(30, 21), (31, 23), (31, 23), (32, 24)], 'IRF':[],
 'MIC':[] }


# Tornem a fer una consulta als IRL
>>> ua.sensor["IRL"]
(36, 25)

>>> ua.sensor.status["IRL"]
[(36, 25), (30, 21), (31, 23), (31, 23)]

>>> ua.sensor.status
{'LDR':[(73, 74), (74, 73), (75, 73)], 'LFW':[],
 'IRL':[(36, 25), (30, 21), (31, 23), (31, 23)], 'IRF':[],
  'MIC':[] }


# Fem una consulta IRF
>>> ua.sensor["IRF"]
(255, 255)

>>> ua.sensor.status["IRF"]
[(255, 255)]

>>> ua.sensor.status
{'LDR':[(73, 74), (74, 73), (75, 73)], 'LFW':[],
```

```
 'IRL':[(30, 21), (31, 23), (31, 23), (32, 24)],
 'IRF':[(255, 255)], 'MIC':[] }

# Obtenim una 5 mostres de só
>>> for i in xrange(5): ua.sensor["MIC"]
(31,)
(31,)
(30,)
(32,)
(31,)

>>> ua.sensor.status["MIC"]
[(31,), (32,), (30,), (31,)]

>>> ua.sensor.status
{'LDR':[(73, 74), (74, 73), (75, 73)], 'LFW':[],
 'IRL':[(30, 21), (31, 23), (31, 23), (32, 24)],
 'IRF':[(255, 255)], 'MIC':[(31,), (32,), (30,), (31,)] }
```

Observem com en el cas del `IRL` tot i que hem fet més de quatre consultes, només es mantenen les últimes quatre últimes mostres rebudes.
Contemplem també que, com en el cas del `LFW`, si no s'han rebut mostres, la llista roman buida.

En algunes ocasions, ens pot interessar emmagatzemar més mostres d'un cert sensor per dur un seguiment més acurat. Per modificar la profunditat de memòria de l'historial d'un sensor cal cridar el mètode `setStepMem()`. Exemple:

```
>>> ua.sensor.status
{'LDR':[], 'LFW':[], 'IRL':[], 'IRF':[], 'MIC':[] }

>>> for i in xrange(8): ua.sensor["LFW"]
(9,)
(9,)
(9,)
(9,)
(12,)
(12,)
(9,)
(12,)

>>> ua.sensor.status["LFW"]
[(12,), (9,), (12,), (12,)]
```

```
# Historial de LFW amb una profunditat de 16 mostres
>>> ua.sensor.setStepMem("LFW", 16)
True

>>> for i in xrange(8): ua.sensor["LFW"]
(14,)
(14,)
(14,)
(14,)
(15,)
(15,)
(14,)
(9,)

>>> ua.sensor.status["LFW"]
[(9,), (14,), (15,), (15,), (14,), (14,), (14,), (14,), (12,),
 (9,), (12,), (12,)]

>>> for i in xrange(6): ua.sensor["LFW"]
(12,)
(12,)
(12,)
(9,)
(9,)
(0,)

>>> ua.sensor.status["LFW"]
[(0,), (9,), (9,), (12,), (12,), (12,), (9,), (14,), (15,),
 (15,), (14,), (14,), (14,), (14,), (12,), (9,)]


# Reduïm la longitud de l'historial de LFW a 8 mostres
>>> ua.sensor.setStepMem("LFW", 8)
True

>>> ua.sensor.status["LFW"]
[(0,), (9,), (9,), (12,), (12,), (12,), (9,), (14,)]
```

4 Actuadors

Els actuadors són tots aquells elements que sense ser els motors, realitzen una acció del robot. En aquesta classificació tenen cabuda els elements tals com: leds, piezo buzzer, micròfon, etc...
El control d'aquests dispositius s'encapsula dins la classe `Misc`, instanciada en `Uabot` com a objecte `misc`:

```
>>> ua.misc
(misc.Misc instance at 0x00c486c0)
```

De la mateixa manera que les classes anteriors, podem consultar quantes comandes enviades i rebudes hem cursat:

```
>>> ua.misc.nReq()
0

>>> ua.misc.nAck()
0
```

La plataforma UABot disposa actualment de sis leds, un piezo buzzer i un micrófon. La numeració dels leds és la següent: $L_5L_4L_3L_2L_1L_0$ on L_5 és el led de més a l'esquerra i L_0 el led de més a la dreta.

4.1 `Misc.touch(msk)`

Activa leds (set) indicats amb la màscara `msk`. `msk` $\in [0, 63]$. Valors més alts de 63 només serà considerat el seu reste de la divisió amb 64 ($msk = msk(mod64)$). Exemple:

```
# Set L_0
>>> ua.misc.touch(0x1)
True

# Set L_4
>>> ua.misc.touch(0x10)
True

# Set L_3 i L_2
>>> ua.misc.touch(0x6)
True

# Set L_0. No passa res
>>> ua.misc.touch(0x1)
True
```

4.2 `Misc.untouch(msk)`

Mètode invers a l'anterior. Apaga un led si amb anterioritat ha estat activat amb `touch()`. Si el led esta apagat, la crida no té cap efecte.

4.3 `Misc.oscLeds(msk, pre)`

4.4 `Misc.led()`

Retorna una llista de 6 elements amb l'estat de cada led.

4.5 `Misc.osc()`

Retorna una llista de 6 elements amb l'estat de cada led oscil·lant. Exemple:

```
# Setegem tots els leds
>>> ua.misc.touch(0x3F)
True

>>> ua.misc.led()
(1, 1, 1, 1, 1, 1)

>>> ua.misc.osc()
(0, 0, 0, 0, 0, 0)

# Leds oscil·lants L_0 i L_5 amb prescaler de (3 + 1) * 20ms
>>> ua.misc.oscLeds(0x21, 3)
True

>>> ua.misc.led()
(0, 1, 1, 1, 1, 0)

>>> ua.misc.osc()
(1, 0, 0, 0, 0, 1)
```

4.6 `Misc.speaker(v, pre)`

Mètode que activa l'speaker. La freqüència de treball ve determinada per els valors `v` i `pre`. `v` ha de ser un valor enter comprés entre 0 i 255 mentre que `pre` hauria de ser enter amb valor de 0 a 5. `v` és un valor numèric de venciment del timer mentre que `pre` és un prescaler (veure la següent taula). Els dos en conjunció determinen el període de la senyal quadrada que té com entrada el piezzo buzzer. Per determinar la freqüència del só resultant,

podem emprar la següent equació:

$$f_{osc} = \frac{16 \cdot 10^6}{2 \cdot N \cdot (1 + \mathbf{v})}$$

on el parametre N de la equació està relacionat amb `pre` de la següent manera:

pre	**N**
0	-
1	1
2	8
3	64
4	256
5	1024

Taula 1: Taula de prescalers

5 Control

Sovint ens pot interessar la consulta de certa informació. A continuació exposem una sèrie de mètodes disponibles per al control d'execució i obtenció d'informació estadística.

5.1 `Uabot.nReq()`

Proporciona el nombre total de consultes efectuades.

```
>>> ua.nReq()
23
```

5.2 `Uabot.nAck()`

Proporciona el nombre total de consultes acceptades.

```
>>> ua.nAck()
20
```

Considerant que hem enviat 23 consultes i només hem rebut 20 confirmacions, llavors 3 peticions no han estat acceptades o trameses correctament.

5.3 Atributs `Uabot.bSnd`, `Uabot.bRcv`

Contabilitzen el nombre de bytes enviats (`bSnd`) i el nombre de bytes rebuts (`bRcv`). Exemple:

```
>>> ua.bSnd
23

>>> ua.bRcv
238
```

Seguint l'exemple, hem enviat 23 bytes i hem rebut 238 bytes en total.

5.4 `Uabot.testTaskMotor()`

Mètode de consulta (ping) a la tasca Motors resident en el microcontrolador. Serveix per verificar la seva existència i funcionament.

```
>>> ua.testTaskMotor()
M-Task: OK
```

5.5 `Uabot.testTaskMotor()`

Mètode de consulta (ping) a la tasca Sensors resident en el microcontrolador. Serveix per verificar la seva existència i funcionament.

```
>>> ua.testTaskMotor()
S-Task: OK
```

5.6 `Uabot.reset()`

Força un reset software al robot. L'operació de reset suposa reiniciar el sistema operatiu de nou. Addicionalment, s'invoquen els mètodes `testTaskMotor()` i `testTaskSensors()` per a la comprovació que ambdues s'executen. Finalment, s'actualitzen a zero totes les variables estadístiques. Exemple:

```
>>> ua.nReq()
23
>>> ua.nAck()
20
>>> ua.bSnd
23
>>> ua.bRcv
238

>>> ua.reset()
M-Task: OK
S-Task: OK

>>> ua.nReq()
0
>>> ua.nAck()
0

>>> ua.bSnd
0
>>> ua.bRcv
0
```

Resum