Humanoides: documentació i refèrencia

Altrament conegut com "els caòtics apunts de l'Aniol passats a net"

Aloma Alenyà, Martí Zaera i Aniol Garcia

Juliol 2017

1 Introducció

Aquest document pretén ser una continuació/ampliació del document humanoids [1] que va fer la Laia Freixas. És un recutll de procediments, informació i documentació del que hem anat fent, així com l'explicació de problemes que hem anat trobant i com solucionar-los.

2 Índex de programes

- *all_movements*: Programa per provar moviments
- *auto_gir*: Programa per fer debug de la brúixola. Fixa una direcció inicial i quan el gires hi torna
- *casal*: demo de caminar recte i parar quan trobi un obstacle al davant. Fet per la visita d'un casal d'infants autistes.
- *do_nothing*: fa la posició inicial (*action_set_page*(31)) i no fa res més.
- *estructura_laberint*: Ara per ara camina cap a l'esquerra fins que troba una paret i en trobar-la torna cap a la dreta fins que en troba una altre.
- *laberint*: Màquina d'estats completa del laberint, amb els comprovadors de les funcions de moviment.
- *laberint_compensating*: Igual que laberint, però utilitzant el bno055 en lloc del compass i utilitzant les funcions compensating.
- *lectura_serial*: Primeres proves de comunicació serial amb el robot.
- *moviments_simples*: camina o gira per sempre segons el botó que es premi.
- *read_compass*: llegeix brúixola. Fer servir *sensor.c* en comptes d'aquest.
- *sensor*: fa lectures de qualsevol dels sensors i les imprimeix per serial
- *turn_angle*: Proves de precisió de la funció *turn_angle*
- *vision_cm*510: Màquina d'estats base pel programa del cm510 de la prova de visió

[1] humanoids.pdf s'hauria de trobar en el mateix directori que aquest fitxer

- *vision_girs_exactes*: Ara per ara és un INTENT de fer girar el robot 45 graus amb una mica de precisió
- *walk_correction*: Fa caminar el robot recte corregint segons el compass (no acaba d'encadenar bé els moviments)
- *walk_correction_compensating*: Caminar recte però utilitzant les funcions compensating i el bno055.
- *walk_forward*: Fan caminar el robot recte sense modificacions
- *walk_prova*: Camina endavant fins que troba un obstacle. LLavors para i encén un llum esperant que l'obstacle desaparegui.

3 Sensors

3.1 Compass

Ara mateix en desús, substituït per bno055, que té més precissió. La brúixola té un rang de $[0, 3600]$, i teòricament cada 10 és un grau. S'inicialitza amb *exp_compass_start*() i es llegeix el valor amb *exp_compass_get_avg_heading*(), que ja retorna una mitjana de diverses lectures seves. No és el sensor més precís del món. **ALERTA**: hi ha funcions que no utilitzen el reang de $[0, 3600]$, sinó que en fan servir un de $[0, 360]$.

3.2 bno055 (substitut de compass)

És un sensor que fa de brúixola, giroscopi i magnetòmetre. La brúixola té un rang de $[-1800, -250] \cup [0, 2050]$, és a dir, que de -250 salta directament a 0 i després aquests 250 els afegeix al final. Hi ha una funció que es diu *bno055_correction* que arregla això donant un rang de $[-1800, 1800]$. Si es vol un rang de $[0, 3600]$ només cal sumar-li 1800 al valor. Tal com passava amb compass, algunes funcions utilitzen aquest rang dividit entre 10. Per inicialitzar-lo s'utilitza *exp_bno055_start*() i per llegir-ne valors *exp_bno055_get_heading*(). **ALERTA**: hi ha funcions com *turn_angle*() que estan programades per fer servir el compass en lloc del bno055. O bé s'ha de modificar la llibreria o bé redefinir la funció en el propi codi amb les ordres correctes.

3.3 IR

Rang de $[82, 500]$ per al de llarga distància (aproximadament). Com més pròxim és l'objecte, més augmenta el valor, encara que a uns 5 o 6 *cm* del sensor el valor torna a decrèixer al apropar l'objecte.

3.4 Gyro

Mesuren velocitat angular en un rang de $[45, 455]$. S'ha de vigilar perquè a vegades el giroscopi que monitoritza l'eix x està connectat al port de l'eix y i al revés. Per defecte l'eix x correspon al port 3 i l'eix y al port 4. Es poden invertir les direccions dels eixos (l'equivalent a rotar el sensor 180 graus amb un paràmetre a

```
cm510_controller_fw/motion/include/motion_cfg.h
```

Si els sensors retornen sempre valors incoherents i fixos, el més probable és que s'haigi de reescriure el firmware de la placa d'expansió. Hi ha documentació del procediment a l'apartat 5.1 del document humanoids.pdf.

4 Simulador

Per utilitzar el simulador cal primer configurar-lo editant una sèrie de fitxers

4.1 CMakeLists.txt

Per accedir al directori d'aquest fitxer cal executar la següent comanda en un terminal:

```
$ roscd cm510_controller_fw
```

En aquest directori hi haurà el fitxer CMakeLists.txt, en el qual s'haurà d'editar la línia

```
SET(ProjectPath ~/directori/del/programa)
```

amb el directori del programa a simular. Unes quantes línies més avall s'ha d'especificar el fitxer *.c*.

4.2 .xacro i .yaml

Si es vol modificar l'estructura del robot dins el simulador cal modificar uns fitxers .xacro i .yaml. El fitxer .xacro conté les declaracions de cadascuna de les peces, sensors i joints, i el fitxer .yaml conté la informació de posició i comportament de cadascun d'aquests elements.
Aquest primer arxiu es diu *bioloid_ceabot.xacro* i el segon *bioloid_ceabot.yaml*. Ambdós arxius són al següent directori:

```
ri-lab/iri_ws/src/bioloid_robot/bioloid_description/urdf/
```

A continuació s'adjunta un exemple de la configuració d'un sensor IR en el fitxer *bioloid_ceabot.yaml*:

```
<xacro:sharp_ir name="nom" parent="part_a_la_qual_s'engaxa" update_rate="20"
    fov="0.05" min_range="0.1" max_range="0.8">
  <origin xyz="(x␣y␣z)" rpy="(x␣y␣z)" />
</xacro:sharp_ir>
```

on $(x\ y\ z)$ és la posició relativa a *parent* i $(r\ p\ y)$ és l'orientació.

4.3 Compilar i executar

Per compilar els programes per utilitzar-los al simulador s'utilitza catkin. El procés de compilació és el següent:

```
$ roscd
$ cd ..
$ catkin_make --only-pkg-with-deps bioloid_controller_cm510
```

Per executar el simulador amb el programa prèviament compilat, només cal executar

```
$ roslaunch bioloid_apps ceabot_base.launch robot:=myrobot mtn_file:=
    ceabot_motions
```

on *ceabot_base.launch* es pot substituir pel fitxer d'entorn (l'escenari) del simulador, a escollir entre

- *ceabot_base.launch*: entorn pla sense cap obstacle.
- *ceabot_obstacles.launch*: entorna amb una reproducció de l'estructura de la prova de laberint del CEABOT. S'ha de tenir en compte que l'estructura no està del tot a escala, i les mides no són les reals que s'utilitzaràn en competició.
- *ceabot_vision.launch*: entorn amb una reproducció de la prova de visió de CEABOT. Tal com l'anterior, l'escala no és del tot exacta.

El paràmetre *robot* indica quins robot s'ha de simular. Si no s'ha creat un robot nou i s'ha modificat el model per defecte, no cal tocar-ho. Finalment, el paràmetre *mtn_file* indica la llibreria de moviments a carregar, i en principi no s'hauria de tocar.

4.4 Particularitats del simulador

S'ha de vigilar en llegir els ports dels sensors al simulador, ja que aquests no necessàriament correspondràn als del robot real, sinó que correspondràn als definits pels fitxers .xacro i .yaml.

5 Moviments

Aquest apartat és posiblement el tema més delicat. Els moviments del robot es fonamenten en les "pàgines", que entenem que són petits conjunts de moviments d'alguns servomotors concrets. Però aquestes pàgines no es criden directament, sinó que a sobre seu hi ha construïda tota una llibreria amb funcions més senziles de moviments, la *mtn_library.c*, que es troba a

```
cm510_controller_fw/motion/src/mtn_library.c
```

Allà hi ha totes les funcions de moviment que es faràn servir.

5.1 Moviment continuat

Quan es crida una funció de moviment, aquesta només fa una iteració. Per exemple: al cridar $walk\_forward()$ el només robot només farà una passa. D'aquesta manera, per obtenir moviment continuat s'ahaurà d'anar cridant les funcions repetidament.

5.2 Aturar moviments

Contràriament al que pugui semblar, per aturar el moviment del robot no és suficient amb cridar la funció $mtn\_lib\_stop\_mtn()$. Si només es crida aqueta funció i es deixa de fer crides a la funció de moviment que s'està intentant parar, el moviment pararà en sec, però no l'acabarà correctament tornant a l'estat inicial. Això provocarà que el següent moviment que s'executi comenci en la posició en que s'ha quedat el robot, fent moviments erronis i imprecisos. El que fa $mtn\_lib\_stop\_mtn()$ és aixecar un flag a la màquina d'estats interna del moviment que s'estava executant, indicant que comenci els moviments de parada.

D'aquesta manera, fins i tot després d'haver donat la instrucció de parar un moviment, aquest s'ha de continuar cridant fins que retorni $0x01$ (que indica que el moviment ha acabat del tot), permetent d'aquesta manera acabar el moviment correctament.

Per tal de veure aquest comportament, a continuació s'adjunta un fragment d'una màquina d'estats que camina recte fins que s'apreta el botó avall i es torna a encendre amb el botó amunt. Generalment l'estructura serà similar a aquesta:

```
case walk:
       //Primer tots els casos que poden fer parar el moviment
       if(is_button_rising_edge(BTN_DOWN)
       {
              mtn_lib_stop_mtn();
       }

       if(walk_forward() == 0x01) //Comprovem si el moviment ha acabat i alhora
            cridem la funcl de moviment
       {
              state = stop; //Canvi d'estat en cas d'haver acabat el moviment
       }
       else
       {
              state = walk; //Encara que hagi rebut l'ordre de parar, es torna
                  a executar tot l'estat fins que el moviment pari.
       }
       break;

case stop:
       if(is_button_rising_edge(BTN_UP))
       {
              state = wait_ready;
       }
       else
```

```
        {
                state = stop;
        }
        break;
```

5.3 Girs

Per tal d'executar els girs correctament és necessari que el balance estigui desactivat. En cas d'estar-ho, s'intentarà compensar els canvis de centre de massa necessaris per girar. Per desactivar-lo només cal cridar la funció *balance_disable_gyro*() abans de girar o bé comprovar si està activat amb *balance_is_gyro_enabled*() i desactivar-lo si és necessari. És important tornar-lo a activar posteriorment abans de tornar a caminar recte. El més recomanable és comprovar si està activat o no i canviar-lo d'estat si és necessari.

S'ha de tenir en compte que en la majoria dels nostres programes el balance s'activa per defecte en el *user_init*.

6 Configuració del robot

6.1 cm510_cfg.h

La majoria de coses parametritzables de la configuració del robot es troben en aquest fitxer, que està a

```
cm510_controller_fw/controllers/include/cm510_cfg.h
```

En aquest fitxer es poden definir, per exemple, els ports del giroscopi. És important que, després de fer canvis en aquest fitxer es torni a fer un make del directori *controller*, és a dir, que s'executin les següents comandes:

```
$ cd ..
$ make clean
$ make
```

6.2 Offsets

Per definir els offsets, cal posar primer la placa *cm*510 en mode slave. Per fer-ho cal definir el robot com a slave a

```
cm510_controller_fw/controller/include/cm510_cfg.h
```

i des de

```
cm510_controller_fw/controller/
```

executar

```
$ make clean
$ make
$ sudo modprobe ftdi_sio
$ make download
```

i des de

```
bioloid_robot_driver/
```

executar

```
$ sudo rmmod ftdi_sio
Encendre el robot i esperar la llumeta verda
$ ./bin/bioloid_offset
```

El robot s'hauria d'aixecar i amb el programa s'haurien de modificar les posicions dels servos segons els valors introduïts en temps real.

Un cop acabat, s'ha de tornar a definir el robot coma a master seguint el mateix procediment per posar-lo com a slave.

Els offsets definits aquí s'emmagatzemen a la EPROM del robot i per tant és de les primeres coses que s'executen en encendre el robot.

6.3 Calibració del bno055

Per utilitzar el sensor bno055, cal un a calibració prèvia. Cada programa que l'utilitza té programada una funció que esborra els valors de l'última calibració de la memòria no volàtil si s'apreta el botó amunt en engegar. La pròxima vegada que s'engegui el robot, el programa esperarà una calibració, i en principi no s'hauria d'encendre el llum verd.

Per calibrar els giroscopis tan sols cal deixar-los uns quants segons sense fer res. Pel compass, cal dibuixar 8 a l'aire amb el sensor, i per l'acceleròmetre cal posar-lo en almenys 6 orientacions diferents uns quants segons. Per l'acceleròmetre és recomenable que les orientacions perpendiculars als eixos X, Y, Z. Un cop fet tot sixò, s'hauria d'encendre la llumeta verda i llavors ja es podria executar el programa.

7 Errors freqüents

Si el robot té problemes d'equilibri, hi ha diverses opcions:

- Que tingueu els giroscopis girats, és a dir, que es facin compensacions a l'eix x segons les lectures del giocopi de l'eix y i al revés. Comprovar primer si és així fent lectures dels dos eixos i en cas d'estar girats modificar els paràmetres a cm510_cfg.h.
- Que el balance estigui activat/desactivat quan no ho hauria d'estar.
- Que el robot no tingui els offsets correctes

- Que hi hagi alguna articulació del robot amb molt de joc. Es recomana comprovar i apretar tots els cargols de tant en tant.

Molts errors del *make* o *make download* (sobretot els que tenen a veure amb $fw\_downloader$) són per culpa dels paths. Reviseu el $makefile$ que siguin correctes.

Si en fer *make download* hi ha errors del tipus

```
Error: [CComm class] - [CRS232 class] - - Impossible to open the serial port.
 - fw_serial_dev
```

o

```
Error: CBioloidRobot error - No FTDI USB devices found
```

solen ser per algun d'aquests motius:

- Teniu el serial/usb desconectat (passa més sovint del que pot semblar)
- Teniu un screen/cuteCom/minicom obert ocupant el port. S'arregla tancant el programa.
- Heu fet *sudo rmmod* $ftdi\_sio$. S'arregla executant *sudo modprobe* $ftdi\_sio$

Si els sensors retornen valors incoherents i fixos, el millor sol ser reescriure el firmware de la placa d'expansió. Hi ha instruccions de com fer-ho a l'apartat 5.1 de humanoids.pdf.

8 Preguntes encara sense resposta

- $walk\_forward\_compensating()$ necessita tenir el balance activat?