Humanoides: documentació i referència

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 recull 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 Contingut del directori

2.1 docs

Directori de la documentació: hi ha tant aquest arxiu com el que va fer la Laia, així com les bases del concurs CEABOT del 2017.

2.2 other_approaches

Directori dels programes que o bé no han funcionat com esperàvem o sí funcionen però no es van utilitzar a la competició. Crec que, encara que alguns estiguin inacabats o no siguin totalment funcionals, és útil tenir-los tant per donar altres idees de com poden funcionar els programes com per saber què no és recomanable fer o què no funciona bé. A l'inici de cada programa hi ha una petita explicació de què és el que es pretenia fer i si ho fa o no.

2.3 programs

Directori dels programes utilitzats en competició. No necessàriament són millors que les seves alternatives de *other_approaches*, però són els que vam creure que donarien millors resultats a la competició.

2.3.1 laberint

Versió del Laberint que millor va funcionar. Camina lateralment fins a trobar un forat suficientment llarg (és a dir, una línia recta cap a la paret del final). Fa unes quantes passes més per

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

assegurar-se que està dins el forat i camina en línia recta compensant fins que troba la paret del final. Fa mitja volta i torna a caminar en línia recta fins a tornar a trobar una paret.

2.3.2 prova_lliure

Bàsicament obre una càmera i busca la cara més gran. La idea era fer que el robot mirés a la cara que es trobava, però no vam tenir temps d'implementar el programa del robot ni de fer la comunicació amb la cm510. Aquest programa és tan sols el que aniria a la BeagleBone i passaria la informació al robot.

2.3.3 stairs

Camina endavant fins que troba l'escala i la puja. ALERTA!: Aquest programa puja escales INFINITAMENT, mai les baixa! A *other_approaches* hi ha altres implementacions que sí fan l'acció de baixar.

2.3.4 sumo

En principi es va movent fent atacs laterals i quan detecta l'enemic ataca cap a aquella direcció

2.3.5 vision_cm510

Programa de la cm510 per la prova de visió. Llegeix el QR descodificat que li envia vision_computer i gira el nombre de graus corresponent.

2.3.6 vision_computer

Programa de la BeagleBone per capturar imatges amb la càmera i descodificar el contingut del QR. També hi ha la part de comunicació serial amb la cm510 per tal de passar la informació.

2.4 telemetry

Directori amb petits programes per llegir sensors del robot per serial. Útils per fer proves i lectures ràpides.

2.5 tests

Els primers programes de proves. Són petits exemples que fan poques accions. Serveixen per controlar que tot funciona correctament i veure com es fan les coses.

2.6 useful

Hi ha backups dels fitxers .xacro i .yaml, així com les pàgines de moviments per defecte i algunes plantilles per programes i Makefiles.

3 Sensors

3.1 Compass

Ara mateix en desús, substituït per bno055, que té més precisió 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. **ALERTA**: hi ha funcions que no utilitzen el rang 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 amb les ordres correctes.

3.3 IR

Rang de $[82, 500]$ (aproximadament) per al de llarga distància. Com més pròxim és l'objecte, més augmenta el valor, encara que a uns 5 o 6 cm del sensor el valor torna a decréixer 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'hagi de reescriure el firmware de la placa d'expansió. El procediment a seguir per tal de reescriure el firmware és el següent:

1. Descarregueu o feu un pull del repositori *bioloid_exp_board_fw*[2]
2. Feu un make dins bioloid_exp_board_fw
3. Connecteu la placa d'expansió a l'ordinador amb un cable USB-MiniUSB i assegureu-vos que reconeix la placa com a ttyUSB0 (podeu veure-ho executant la comanda *dmesg*). Si no us el reconeix com a tal, probablement teniu connectat algun altre dispositiu USB. Desconnecteu-lo i torneu a connectar la placa d'expansió.
4. Enceneu el robot i executeu *make download* abans d'un segon

4 Configuració del robot

4.1 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
```

4.2 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 :
// CM510 controller operation mode
//#define CM510_master
#define CM510_slave
```

i des de

```
cm510_controller_fw/controller/
```

executar

```
$ make clean
$ make
```

[2] https://gitlab.iri.upc.edu/humanoides/bioloid_exp_board_fw

Ara cal pujar el programa slave, que es troba a la carpeta d'exemples de $cm510\_controller\_fw$, és a dir a

```
cm510_controller_fw/examples/slave
```

Allà cal fer

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

Finalment 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 que per posar-lo com a slave.

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

4.3 Pàgines de moviments

La controladora té un conjunt de pàgines de moviments que defineixen les posicions dels motors en els diversos passos de cada moviment. A vegades interessa tenir un conjunt de funcions de moviment (per exemple: uns moviments per al sumo i uns altres per pujar escales), i hem de canviar el fitxer de pàgines que té el robot. Això es fa amb una eina anomenada $mtn\_downloader$. Per pujar un arxiu *.mtn* al robot fem:

```
$ mtn_downloader -d <port_serie_dispositiu> -m <fitxer_mtn>
```

Per exemple:

```
$ mtn_downloader -d /dev/ttyUSB0 -m sumo.mtn
```

Observacions: Els fitxers *.mtn* tenen un nombre fix de pàgines i pot ser que moltes d'aquestes estiguin buides. Així, quan es vol afegir una pàgina, se n'ha d'esborrar una d'aquestes buides i posar-hi en el seu lloc la nova. Un altre fet important és que les pàgines es criden per la seva ID. La ID d'una pàgina correspon a la posició que ocupa dins el fitxer de pàgines. El recomanable és buscar la ID de l'última pàgina plena i posar les noves a continuació, de manera que sigui fàcil saber el seu ID. No les poseu mai davant, ja que si no totes les ID de les altres pàgines es veuran afectades! Posteriorment es pot fer un define i assignar un nom a cada ID.

4.4 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 recomanable que les orientacions perpendiculars als eixos X, Y, Z. Un cop fet tot això, s'hauria d'encendre la llumeta verda i llavors ja es podria executar el programa.

El problema és quan havent seguit tots els passos no s'encén la llumeta. No podem saber quin pas ha fallat, per tant s'ha de tornar a fer tota la calibració fins que s'encengui.

5 Moviments

Aquest apartat és possiblement el més delicat. Les pàgines de moviments ja esmentades no es criden directament, sinó que a sobre seu hi ha construïda tota una llibreria amb funcions més senzilles 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 faran servir.

5.1 Moviment continuat

Quan es crida una funció de moviment, aquesta només fa una iteració. Per exemple: al cridar $walk\_forward()$ el robot només farà una passa. D'aquesta manera, per obtenir moviment continuat s'haurà 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 aquesta 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 realment 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 finalitzar 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 es prem 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 funció 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 actiu, 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 a *user_init*.

6 Simulador

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

6.1 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*.

6.2 .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.

6.3 Compilar i executar

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

```
$ roscd bioloid_controller_cm510
```

Aquí hem d'editar el document CMakeLists.txt i posar el path al nostre programa, així com el nom del programa. Buscar i editar les línies

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

i

```
# main application module
${ProjectPath}/programa.c
```

Finalment cal fer

```
$ 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'utilitzaran en competició.
- *ceabot_vision.launch*: entorn amb una reproducció de la prova de visió de CEABOT. Tal com a 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.

6.4 Particularitats del simulador

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

7 Compilar i pujar

Per tal de compilar i pujar un programa a la controladora s'utilitzen els makefiles. Si feu *make* en el directori on hi ha el codi en C i el seu corresponent makefile es generaran els fitxers .map, .elf, .hex i .o. Un cop generats, al fer *make download* s'executarà automàticament el $fw\_downloader$ que pujarà el fitxer .hex a la controladora. Així, la seqüència d'ordres a executar és la següent:

```
$ make clean
$ make
$ make download
```

8 Errors freqüents

Si el robot té problemes d'equilibri, hi ha diversos possibles problemes:

- Que tingueu els giroscopis girats, és a dir, que es facin compensacions a l'eix x segons les lectures del giroscopi 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 del makefile. Per exemple, depenent de l'ordinador que utilitzeu l'usuari es dirà *humanoids* o *humanoides*, cosa que s'ha de tenir en compte al escriure els paths absoluts. En alguns makefiles també hi ha paths relatius, que variaran segons on tingueu el makefile i codi. Reviseu-los i comproveu que siguin correctes i s'adaptin a la vostra estructura de directoris.

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 desconnectat (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*

9 Preguntes encara sense resposta

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