progetto unibo.basicrobot24 *unibo.basicrobot24*

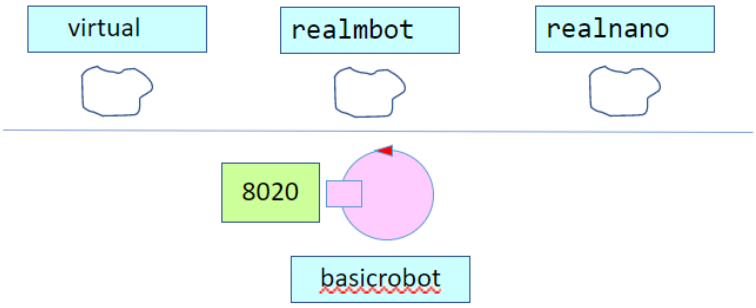
Obiettivo: introdurre un componente software che esegue comandi di spostamento di un DDR-robot ed altre funzionalità utili.

Il servizio offre anche un modo di operare in modo indipendente dalla tecnologia con cui è realizzato uno specifico DDR robot (virtuale o reale).

BR24 requisito tipidirobot

Facciamo riferimento ad almeno tre diversi tipi di robot:

- *VirtualRobot23*
- *Mbot* (realmbot)
- *NanoRobot* (realnano)



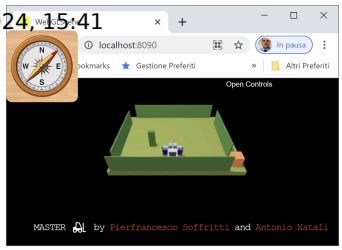
BR24 Una prima architettura

Si delinea una architettura come quella anticipata in *RobotCleaner* e qui riportata:

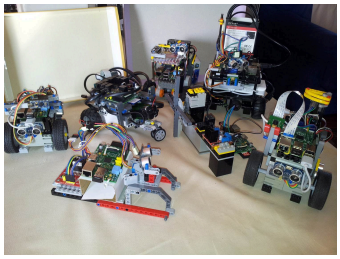
 *images/basicrobotproject.PNG*

BR24: configurazione

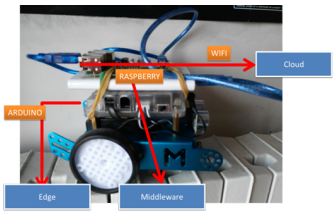
La utility class *robotSupport.kt* si occupa dei dettagli tecnologici specifici di ogni *tipo di robot* utilizzando un supporto diverso per ciascun tipo.



per il VirtualRobot: *virtualrobotSupport*



per il NanoRobot: *nanoSupport*, *motors*



per Mbot: *mbotSupport*

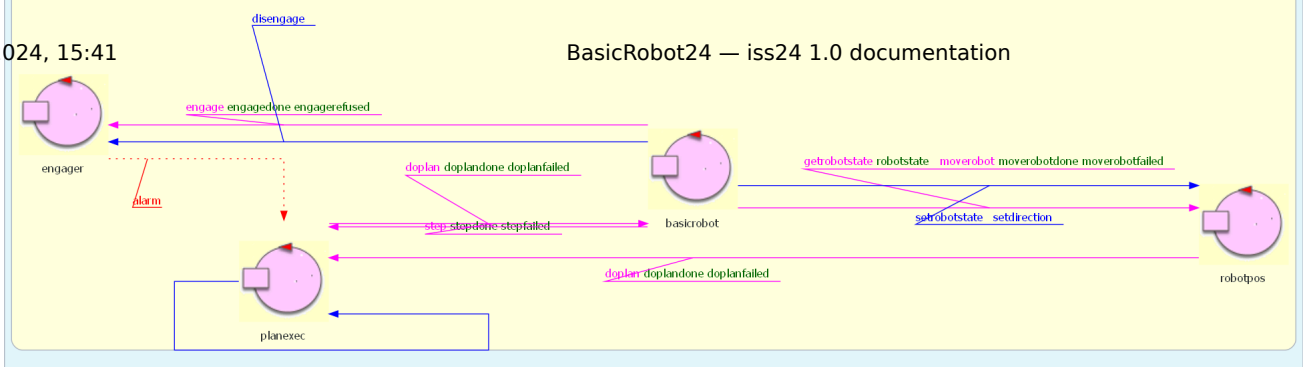
BR24 File di configurazione

Il file di configurazione è impostato su frasi JSon, come ad esempio il seguente **basicrobotConfig.json**:

```
{"type":"virtual","port":"8090","ipvirtualrobot":"..."}
{"type":"realnano","port":"8020","ipvirtualrobot":"dontcare"}
//Arduino connesso al Raspberry:
{"type":"realmbot", "port":"/dev/ttyUSB0", "ipvirtualrobot":"-"}
//Arduino connesso al PC:
{"type":"realmbot","port":"COM6","ipvirtualrobot":"dontcare"}
```

Il progetto unibo.basicrobot24

Il codice è contenuto nel progetto *unibo.basicrobot24*, che fornisce il modello eseguibile *basicrobotqak*, che include i componenti riportati nella figura (generata) che segue:



Componenti (attori):

- *basicrobot*: esegue i comandi di spostamento e altre funzionalità
- *engager*: gestisce l'ingaggio del robot
- *planexec*: esegue piani di movimento
- *robotpos*: realizza il posizionamento del robot in una cella data della mappa

I progetti di riferimento:

- *unibo.basicrobot24*
-

BR24 Le funzioni

Il componente *basicrobot* viene visto dall'esterno come un servizio che realizza un insieme di funzionalità:

1. Esecuzione di richieste di ingaggio: messaggio `Request engage:engage(CALLER)`
2. Esecuzione di comandi elementari di movimento: messaggio `Dispatch cmd:cmd(MOVE)`
3. Esecuzione di step (movimento in avanti per un tempo dato): messaggio `Request step:step(TIME)`
4. Esecuzione di sequenze di movimento (piani): messaggio `Request doplan:doplan(PATH,STEPTIME)`
5. Esecuzione di posizionamento: messaggio `Request moverobot:moverobot(TARGETX, TARGETY)`

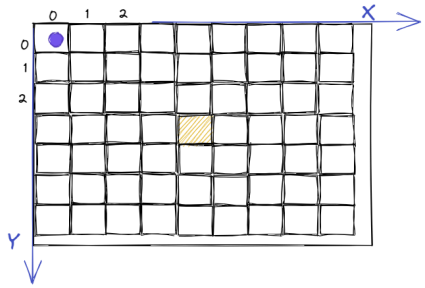
Il *basicrobot* esegue le funzioni 2-5 solo dopo essere stato ingaggiato da chi lo vuole utilizzare.

Per agevolare il testing, un utlizzatore (di solito una console) di nome `gui23xyz9526` viene sempre ammesso all'uso del robot.

- delega engage/disengage al componente *engager*, che utlizza un POJO di classe *OwnerManager*
- delega doplan al componente *planexec*, che utlizza *OwnerManager*
- delega moverobot al componente *robotpos*, che utilizza *Planner23Util* e una mappa della stanza

BR24 e le mappatura dell'ambiente

Il concetto di posizione viene formalizato introducendo una coppia di coordinate cartesiane che identifica una cella della mappa.



Muovere il robot con mossa **step(T)** con tempo **T** tale da spostare il robot (con velocità prefissata) di uno spazio **D**, permette di costruire una mappa della stanza formata da celle quadrate **DxD**. Ad esempio:

```
  0  1  2  3  4  5  6  7  x
0 |r, 1, 1, 1, 1, 1, 1,
1 |1, 1, 1, 1, X, X, 1,
2 |1, 1, 1, 1, X, X, 1,
3 |1, 1, X, 1, 1, 1, 1,
4 |1, 1, 1, 1, 1, 1, 1,
5 |X, X, X, X, X, X, X,
y
```

- **0** denota una cella mai percorsa
- **1** denota una cella libreria
- **X** denota una cella occupata da un ostacolo
- **r** denota la posizione corrente del robot

Un modo per costruire dinamicamente una sequenza di mosse con cui il robot può muoversi dalla posizione corrente **r** a un'altra posizione (libera) sulla mappa, consiste nell'*utilizzo del pianificatore*.

BR24 Componenti

engager	Gestisce i messaggi engage/disengage usando un oggetto Kotlin OwnerManager. La richiesta engage produce la risposta engagedone in caso il robot sia libero e engagerefused nel caso di robot già ingaggiato.

---

planexec	messe, inviando comandi al *basicrobot* e invia la risposta doplandone in caso di successo. Nel caso di ostacolo o di evento alarm, interrompe la esecuzione della sequenza, inviando la risposta doplanfailed. Non usa (e quindi non aggiorna) alcuna mappa della stanza.
robotpos	Gestisce richieste di posizionamento moverobot. Facendo *Uso del planner*, determina un piano di movimento per raggiungere la destinazione, che esegue usando *planexec*. Invia la risposta moverobotdone in caso di successo oppure moverobotfailed se la esecuzione del piano fallisce o viene interrotta. Gestisce anche il comando di utilità Dispatch setrobotstate:setpos(X,Y,D) per allienare la rappresentazione del planner sulla posizione corrente reale del robot fissata manualmente.
basicrobot	Gestisce direttamente comandi elementari cmd e richieste step (inviando la risposta stepdone in caso di successo e stepfailed in caso di ostacolo).

BR24 messaggi

```
System BR24

Dispatch cmd        : cmd(MOVE)       //MOVE=w|s|d|a|r|l|h
Dispatch end        : end(ARG)

Request step       : step(TIME)
Reply stepdone     : stepdone(V)
Reply stepfailed   : stepfailed(DURATION, CAUSE)

Event  sonardata   : sonar( DISTANCE )        //percepito da sonarobs/engager
Event obstacle     : obstacle(X)
Event info         : info(X)

Request  doplan   : doplan( PATH, STEPTIME )
Reply doplandone  : doplandone( ARG )
Reply doplanfailed  : doplanfailed( ARG )

Dispatch setrobotstate: setpos(X,Y,D)
Dispatch setdirection : dir( D )  //D =up|down!left|right

Request engage        : engage(CALLER)
Reply   engagedone    : engagedone(ARG)
Reply   engagerefused : engagerefused(ARG)

Dispatch disengage    : disengage(ARG)

Event alarm           : alarm(X)
Dispatch nextmove     : nextmove(M)
Dispatch nomoremove   : nomoremove(M)
```

```
Reply moverobotdone  :  moverobotOK(ARG)
Reply moverobotfailed:  moverobotfailed(PLANDONE, PLANTODO)
```

BR24 come esecutore di movimenti

- Il robot è un oggetto di dimensioni finite, inscrivibile in un cerchio di diametro **D** (**unità robotica**) ed esegue movimenti a velocità costante.
- Il *basicrobot* fornito dal committente è un puro esecutore di comandi, con cui il robot può effettuare singole mosse o sequenze di mosse, a seguito di messaggi di richiesta.

Dispatch **cmd:cmd(M)**	Il robot cerca di eseguire la mossa **M (w|s|a|d|l|r|h)**
Request **step:step(T)**	Il robot cerca di fare un movimento in avanti di durata **T** e risponde con: ◦ Reply **stepdone:stepdone(ok)** in caso di successo ◦ Reply **stepfailed:stepfailed(DURATION,CAUSE)** in caso di fallimento
Request **doplan:doplan(PLAN,OWNER,STEPTIME)**	Il robot cerca di eseguire (via *planexec*) la sequenza di mosse **PLAN** e risponde con: ◦ Reply **doplandone:doplandone(ok)** in caso di successo ◦ Reply **doplanfailed:doplanfailed(PLANTODO)** in caso di fallimento

BR24 come sorgente di informazioni

Il robot è una risorsa osservabile che emette informazioni sullo stato del suo ambiente. Nel caso del virtualrobot viene resa disponibile informazione relativa ai dati rilevati dai sonar presenti nella stanza

La esecuzione di un piano

BR24 planexec

L'attore *planexec* ha il compito di eseguire un piano definito come una sequenza di mosse.

BR24 Sequenza di mosse in forma verbosa/compatta

La sequenza di mosse che definisce un piano può essere espressa in **forma verbosa** (ad esempio **[w, w, l, w, w]**) oppure in **forma compatta** (ad esempio **wwlww**).

Il messaggio di richiesta di esecuzione di un piano ha la forma:

```
Request doplan:doplan(PLAN,OWNER,STEPTIME)
```

- *PLAN*: la sequenza di mosse da eseguire, espressa in *forma verbosa* oppure in *forma compatta* ;
- *OWNER*: il nome dell'attore chiamante (che ha ingaggiato il *basicrobot*);
- *STEPTIME*: il tempo (in msec) di esecuzione dello *step* da parte del *basicrobot*.

A questa richiesta, il *planexec* risponde con due possibili messaggi di risposta:

```
Reply doplandone    : doplandone( ARG )
Reply doplanfailed  : doplanfailed( ARG )
```

- Nel caso di successo (**doplandone**) l'argomento **ARG** del payload della reply non è significativo. In futuro potrebbe rappresentare il tempo di esecuzione.
- Nel caso di fallimento (**doplanfailed**) l'argomento **ARG** del payload della reply rappresenta la porzione di PLAN che è stata eseguita prima del fallimento (per allarme o per l'incontro di un ostacolo).

BR24 Esperimenti di uso (con VirtualRobot)

1. Posizionarsi sulla direcory `unibo.basicrobot24`
2. Attivare il VirtualRobot con il comando `docker-compose -f virtualRobot23.yaml up`
3. Attivare il basic robot on il comando `gradlew run`
4. Aprire la pagina HTML *MapGui.html* in *planusage24*, che offre una GUI che usa MQTT per inviare comandi al basicrobot e per ricevere informazioni di stato sia dal basicrobot sia dal planner
5. Attivare *mapobstaclesrobot.qak* in *planusage24* che usa il *Planner* per esplorazione in colonne vertcali
6. Altro esprimento con *mapobstaclesplan.qak* in *planusage24*, che cerca di raggiungere una cella fuori dalla mappa

BR24 Facade

Il basicrobot può anche essere usato mediante un Browser collegandosi alla porta 8085. si veda *RobotFacade24*

BR24 Uso di docker

BR24 Attivazione da docker.io

Eseguire **docker-compose -f basicrobot24.yaml up** con riferimento alle immagini di **docker.io/natbodocker**

BR24 Creazione immagine docker locale

1. Impostare il file **basicRobotConfig.json** come segue

```
{"type":"virtual", "port":"8090", "iprobot":"IPADDR", "commtrace": "false"}
```

IPADDR deve essere fissato al all'IP del computer su cui viene attivato il virtual robot.

Per modificare il file **basicRobotConfig.json** entro un container **XYZK** senza rifare l'immagine, usare il comando

1. Eseguire **gradlew distTar**
2. Eseguire **docker build -t basicrobot24:1.0 .** Notare il punto finale
3. Eseguire

```
docker run -it --rm --name basicrobot24 -p8020:8020/tcp -p8020:8020/udp  --privileged basi
```

BR24 Creazione file yaml per docker composer

1. Impostare il file **basicRobotConfig.json** come segue

```
{"type":"virtual", "port":"8090", "iprobot":"wenv", "commtrace": "false"}
```

2. Ricreare l'immagine come descritto in *BR24 Creazione immagine docker locale* partendo dal punto **2**.

3. Impostare il file basicrobot24.yaml

```
version: '3'
services:
  wenv:
    image: docker.io/natbodocker/virtualrobotdisi23:1.0
    ports:
      - 8090:8090
      - 8091:8091/tcp
      - 8091:8091/udp
  basicrobot24:
    #image: docker.io/natbodocker/basicrobot24:1.0  #VERSIONE FINALE
    image: basicrobot24:1.0
    ports:
      - 8020:8020/tcp
      - 8020:8020/udp
    depends_on:
      - wenv
```

4. Eseguire **docker-compose -f basicrobot24.yaml up**

BR24 NanoRobot

Si veda:

- *NanoRobot*
- Il file *basicrobotConfigNano.json* (da copiare su Raspberry e ridenominare in **basicrobotConfig.json**)

- Il codice **/home/pi/nat/unibo.basicrobot23-2.0** su RaspberryPi per l'applicazione dello scorso anno
- Il codice **/home/pi/nat/unibo.basicrobot24** su RaspberryPi