Robolang

Magí Toneu, Víctor Massagué i Pau Argelaguet

Facultat d'Informàtica de Barcelona, Universitat Politècnica de Catalunya

Abstract

Robolang és un llenguatge de programació pensat per a facilitar la programació dels robots Lego Mindstorms[1]. Concretament, ha estat dissenyat per al model NXT, però la seva naturalesa el fa fàcilment extensible a altres models de la marca.

Per aconseguir això, Robolang incorpora una sintaxi simple i certes característiques similars a les dels llenguatges de *scripting*, per tal de disposar d'un entorn de desenvolupament àgil i ràpid. A més, també inclou comandes predefinides per a accions comunes del robot, que altrament comportarien un número significatiu de línies de codi.

Finalment, Robolang inclou un compilador que, agafant com a entrada el programa, el compila i el volca al robot automàticament (sempre que aquest estigui connectat via USB o Bluetooth).

[1] https://www.lego.com/es-es/mindstorms

1. Definició del llenguatge

1.1. Descripció general

Robolang és un llenguatge interpretat per una màquina virtual[2], de tipat fort i dinàmic. Els fitxers que contenen codi Robolang són fitxers de text pla amb l'extensió `.rl`.

L'execució del codi és totalment seqüencial, començant per la primera línia del fitxer de text i acabant per la última en estricte ordre d'aparició. No hi ha cap funció `main` ni similars[3].

El format del codi està inspirat en els llenguatges com C/C++: els blocs estan limitats per claudàtors (`{, }`), les instruccions han d'acabar en punt i coma (`;`) i els comentaris poden ser d'una única línia (`// comentari`) o de múltiples (`/* comentari */`).

Com es comenta més endavant en els tipus de dades, no tenim tipus de dades definits explícitament, i això inclou les funcions. Per a definir-les, es fa servir la paraula clau `def` (d'una manera similar a com ho fa Python) i no s'especifica de tipus de retorn. En el cas que una funció no contingui cap expressió `return`, es considera que no retorna res (similar a una funció `void` de C).

Les estructures de control de flux també són molt similars als llenguatges anteriorment mencionat. Tenim els habituals `if, elif, else` i bucles del tipus `while`. També suportem els habituals operadors de comparació

[2]Tot i ser interpretat, al córrer sobre la JVM el *bytecode* generat és molt eficient i a més disposem d'un JIT.

[3]Una funció definida com a *main* es comporta com qualsevol altra funció.

`<, >, <=, >=, ==, !=`. Similarment, conté les operacions bàsiques en reals (suma, resta, multiplicació, divisió i mòdul).

Per accedir a sensors i motors, fem servir el símbol del dòlar (`$A, $1`). L'expressió ens retornarà una referència als ports (A, B, C, 1, 2, 3, 4) per a efectuar operacions directament a un motor o a un sensor.

A nivell de robot, a més, també l'accés als botons que es troben al cos de la CPU, a l'emissor de so del mateix (per emetre beeps i similars) i a la pantalla per a escriure-hi text com a sortida estàndard.

El llenguatge també suporta projectes multi-arxiu mitjançant l'ús de `import`. Quan a l'arxiu principal hi hagi una sentència d'importació, s'inserirà el codi de l'arxiu desitjat per a poder usar-ne les funcions. Com a decisió de disseny, només s'importen les funcions i s'ignora tot el codi seqüencial que no es trobi dins una d'aquestes, ja que aquesta funcionalitat està pensada bàsicament per al desenvolupament de llibreries.

Finalment, Robolang és capaç de generar en un arxiu d'imatge (png) l'arbre de parsing que s'ha fet servir per a interpretar un arxiu en concret.

El desenvolupament d'aquest projecte ha estat completament des de zero i no s'ha fet servir com a base ASL, per la raó de no haver d'estar compromesos per decisions de disseny que no han estat preses específicament per a un llenguatge d'un robot.

1.2. Tipus de dades i estructures

A Robolang, el tipus de les variables no es defineix explícitament, sinó que el compilador és capaç d'inferir-lo. Això no vol dir que no existeixin, és a dir, sí que hi ha comprovació de tipus (inferit) quan es realitzen operacions entre variables o crides a funcions.

Els tipus i les estructures suportades són relativament pocs, ja que s'ha optat per la simplicitat del llenguatge reduint-los als estrictament necessaris per a operar el robot. Són els següents:

- **Cadenes de text**. Conjunt de caràcters alfanumèrics, com `string` a la majoria de llenguatges.
- **Enters**. Nombres enters, com `int` a la majoria de llenguatges. La seva longitud depèn de la plataforma on s'estigui executant, en el cas del NXT, 32 bits.
- **Reals**. Nombres enters de doble precisió, com `double` a la majoria de llenguatges. Noti's que no tenim nombres de coma flotant de precisió simple.
- **Booleans**. Valor cert (`true`) o fals (`false`). En el cas de ser avaluats en una expressió numèrica, prenen el valor de `1` i `0` respectivament.

1.3. Configuració per defecte del robot

Una de les bases de disseny d'aquest llenguatge és el de fer-lo el més genèric possible. Per tant, el llenguatge ha de ser capaç d'adaptar-se al màxim nombre possible de configuracions físiques (sensors, motors, peces i col·locació de tots aquests) que desitgi el programador.

D'altra banda, però, també és objectiu el fet de simplificar el desenvolupament i per tant proveir al desenvolupador d'opcions per defecte que li facilitin les configuracions inicials.

Per aconseguir un compromís entre aquests dos factors, s'ha optat per assumir una configuració per defecte, tal i com es mostra a la figura 1, on hi

ha connectats tres motors (dos per a moció i un per a disparar pilotes), dos sensors de tacte, un sensor de color i un sensor ultrasònic. A més, assumim que aquests estan connectats als ports A, B, C, 1, 2, 3 i 4 respectivament.

A cada programa, es poden executar unes instruccions especials, `setDefault`, que permeten explicitar a quin port està connectat el motor dret, el sensor de tacte, l'ultrasònic...

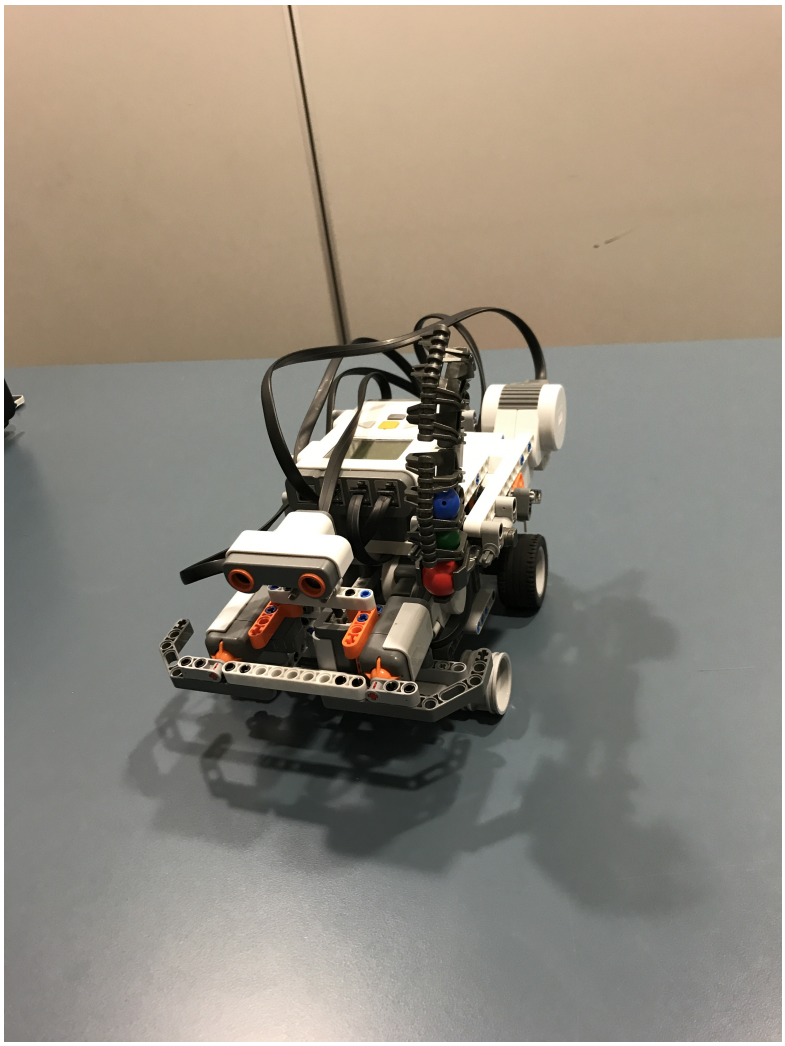

Figura 1: Configuració física del robot usada per defecte

2. Arquitectura

Robolang consisteix d'un compilador que genera codi Java, que al seu torn és compilat a bytecode per a ser executat a la màquina virtual que corre al robot.

Per defecte, Lego inclou un *firmware* poc modificable i només programable de manera visual, per tant, s'ha optat per substituir aquest i instal·lar a la unitat de proves leJOS[4], que és essencialment una JVM modificada que exposa unes APIs que donen accés als components del robot.

El procés de compilació comença amb el parsing dels arxius de text pla en format `.rl`. Mitjançant ANTLR[5] (en la seva versió 3), s'ha definit un lèxic i una sintaxi que generen un arbre.

Llavors, el compilador agafa aquest arbre i el transpila en codi Java, amb l'ajuda de la llibreria Java Poet[6], que facilita molt la feina repetitiva en la generació de codi. El codi generat és compatible amb Java 6[7] i fa servir les ja mencionades APIs de leJOS.

Addicionalment, es dóna un format (identacions, espais, salts de línia) estàndard. De fet, el format en el que es genera el codi segueix les guies d'estil de Java de Google, i es realitza usant una llibreria que posa a disposició la pròpia companyia[8].

---

[4]http://www.lejos.org

[5]http://www.antlr3.org

[6]https://github.com/square/javapoet

[7]La versió que corre al robot, hi va haver problemes en aquest sentit perquè es van començar a implementar funcions en Java 8. El compilador en sí (codi que corre al PC) sí que està fet en Java 8, per aprofitar els avantatges que ofereix aquesta nova versió.

[8]https://github.com/google/google-java-format

Finalment, emprant el compilador (`njxc`) i l'enllaçador (`nxjlink`) proporcionats per leJOS, es compila (es generen diversos arxius `.class`) i s'enllaça (s'empaqueten aquests arxius en un `.nxj`, similar a un `.jar`) per ser llavors pujat i executat al robot mitjançant `nxjupload`.

En aquest procés de compilació, també es compilen les funcions predefinides. Aquestes han estat desenvolupades directament en Java, i quan es compila el codi, es detecta si són cridades i se'ls hi afegeix un `Common.` al davant. A l'estar dins del mateix paquet i ser funcions estàtiques, es poden cridar sempre que el programador ho desitgi. S'ha optat per aquesta alternativa pel fet de la netedat i la claredat que ofereix Java (i les eines del seu entorn) per a desenvolupar funcions més complexes, en un fitxer separat.

Robolang pot ser vist doncs, com una capa d'abstracció per sobre Java i leJOS, ja que permet escriure programes d'una manera molt més ràpida i eficient gràcies a les seves múltiples configuracions per defecte i les funcions predefinides. Com que al final s'està generant codi Java, és possible escriure extensions de Robolang en aquest llenguatge.

El llenguatge està elaborat com a un projecte Maven[9], el gestor de paquets recomanat per a projectes grans elaborats amb ANTLR[10]. S'ha optat per aquesta alternativa per la raó que gestiona dependències, compila i empaqueta els fitxers d'una manera molt més automàtica, ràpida i còmoda.

Tots els passos exposats anteriorment són realitzables un per un i per separat (bàsicament pot ser útil en processos de debug), però també s'inclou un script en Bash que permet executar-los tots de manera sistemàtica, fent

[9] https://maven.apache.org

[10] https://theantlrguy.atlassian.net/wiki/display/ANTLR3/Building+ANTLR+Projects+with+Maven

molt menys tediós el procés.

3. Funcions predefinides

Una part fonamental del llenguatge són les funcions predefinides, que permeten fer interaccions habituals amb el robot d'una manera molt més ràpida i eficient.

Seguidament, exposem totes les funcions predefinides que s'han implementat, juntament amb els paràmetres amb els que han de ser cridades i una breu descripció de la seva funcionalitat.

- `print(val)`, mostra per pantalla el valor de la variable val.
- `clearDisplay()`, neteja la pantalla LCD del robot.
- `forward()`, el robot es mou endavant mitjançant els dos motors per defecte o els definits per l'usuari.
- `backward()`, el robot es mou endarrere mitjançant els dos motors per defecte o els definits per l'usuari.
- `stop()`, si el robot està en moviment, es para.
- `$MotorPort.stopMotor()`, si el motor que està connectat en el port `PortMotor` està en moviment, es para.
- `moveFront(units)`, el robot es mou `units` endavant mitjançant els dos motors definits per defecte o per l'usuari.
- `moveBack(units)`, el robot es mou `units` endarrere mitjançant els dos motors definits per defecte o per l'usuari.

- `$MotorPort.move(bool)`, si `bool` és `true`, el robot es mou endavant, altrament endarrere.
- `rotateLeft(degrees)`, el robot gira cap a l'esquerra `degrees` graus mitjançant els dos motors definits per defecte o per l'usuari.
- `rotateRight(degrees)`, el robot gira cap a la dreta `degrees` graus mitjançant els dos motors definits per defecte o per l'usuari.
- `$MotorPort.rotationMove(units)`, el motor connectat a port `PortMotor` gira `units` rotacions.
- `$Button.waitToBePressed`, pausa l'execució del programa fins que es prem el botó `Button`.
- `explore()`, el robot avança en línia recta fins que xoca amb algun objecte, detectat mitjançant sensors de tacte. Llavors, gira sobre si mateix fins a trobar, mitjançant el sensor ultrasònic, una direcció en la que pot avançar almenys 25cm, i repeteix el mateix procés fins que el botó del robot és premut.
- `exploreUltrasonic()`, similar a `explore`, però ignora els sensors de tacte i es basa només en les distàncies obtingudes pel sensor ultrasònic.
- `followLine()`, el robot segueix una línia de color, diferent del color terra, mitjançant l'ús de `Color Sensor`. El robot gira fins trobar la part esquerre de la línia i llavors amb la altra roda corregeix la direcció per avançar.

- `followLine2()`, mètode en el qual el robot segueix una línia de color a una velocitat més ràpida que `followLine()`. El robot en aquest cas avança recte fins llegir el color del terra, llavors corregeix la seva direcció buscant la línia.
- `shoot(n)`, el robot mitjançant el motor assignat per defecte o definit per l'usuari prèviament, dispara `n` boles.
- `party()`, el robot fa un seguit de rotacions alternades amb tot tipus de sons.
- `waitForPress()`, mètode per pausar el programa fins que l'usuari premi qualsevol botó.
- `$MotorPort.setSpeed(speed)`, assigna al motor del port `MotorPort` la velocitat `speed`.
- `$MotorPort.setAcceleration(acc)`, assigna al motor del port `MotorPort` l'acceleració `acc`.
- `beep()`, el robot genera un so curt.
- `beepSequence()`, el robot genera una seqüencia de sons curts.
- `buzz()`, el robot genera un so llarg.
- `$SensorPort.getDistance()`, retorna la distància al obstacle més proper proporcionada pel `Ultrasonic Sensor` connectat al port `SensorPort`.
- `$SensorPort.isPressed()`, retorna si el `Touch Sensor` connectat al port `SensorPort` està premut.

- `$Button.isPressed()`, retorna si el botó `Button` està premut.
- `$MotorPort.getSpeed()`, retorna la velocitat actual del motor connectat al port `MotorPort`.
- `$MotorPort.getAcceleration()`, retorna la acceleració actual del motor connectat al port `MotorPort`.
- `$MotorPort.getMaxSpeed()`, retorna la velocitat màxima del motor connectat al port `MotorPort`.
- `$SensorPort.getColor()`, retorna el color llegit pel sensor `Color Sensor` connectat al port `SensorPort`.
- `$SensorPort.calibrateWhiteColor()`, actualitza el color blanc del `Color Sensor` connectat al port `SensorPort`.
- `$SensorPort.calibrateBlackColor()`, actualitza el color negre del `Color Sensor` connectat al port `SensorPort`.
- `$SensorPort.getHighValue()`, retorna el color establert com a blanc del `Color Sensor` connectat al port `SensorPort`.
- `$SensorPort.getLowValue()`, retorna el color establert com a negre del `Color Sensor` connectat al port `SensorPort`.
- `delay(time)`, el programa es pausa durant `time` mil·lisegons.
- `setDefault("Component", "Port")`, serveix per canviar la configuració predeterminada del robot. `Component` pot prendre els valors `leftMotor, rightMotor, shootMotor, touchSensor1, touchSensor2`

`ultrasonicSensor, colorSensor`, i `Port` és el nom del nou port on el `Component` és connectat.

4. Exemples de codi

En aquest apartat s'exploren diferents exemples de codi que il·lustren les funcionalitats principals del llenguatge.

4.1. Follow Line

El següent programa defineix una funció que permet que el robot segueixi una línia, al igual que fa la funció predefinida followLine(), però en aquest programa no es fa a partir de la crida a la funció predefinida, sinó que està implementat amb operacions pròpies de Robolang. A més, es fa ús de la funció `setDefault` per establir quins són els ports en els que estan connectats els motors dret i esquerre, i també per establir a quin port estar el sensor de color.

En aquest codi podem veure diferents funcionalitats del llenguatge: definició de funcions (`def function()`), ús de funcions predefinides (`waitForPress()`, `print()`, `clearDisplay()`...), funcions pròpies dels sensors, motors i botons (`$S2.calibrateWhiteColor()`, `$A.setAcceleration()`,`$BESCAPE.isPressed()`...) i estructures de control de flux.

```
def follow(){
setDefault("rightMotor", "B");
    setDefault("leftMotor", "A");
    setDefault("ultrasonicSensor","S2");
    print("calibrate white color");
    print("Press any button to Continue");
    waitForPress();
    $S2.calibrateWhiteColor();
```

```
    clearDisplay();
    print("calibrate black color");
    print("Press any button to Continue");
    waitForPress();
    $S2.calibrateBlackColor();
    print("Put the robot on the left side");
    waitForPress();
    clearDisplay();
    $A.setAcceleration($A.getAcceleration()/4);
    $B.setAcceleration($B.getAcceleration()/4);
    $A.setSpeed($A.getSpeed()/2);
    $B.setSpeed($B.getSpeed()/2);

    while(not($BESCAPE.isPressed())){
        if(($S2.getColor() + 70> $S2.getHighValue() and
        $S2.getColor() -70 < $S2.getHighValue())){
            $B.stopMotor();
            $A.move(true);
        }
        else {
            $A.stopMotor();
            $B.move(true);
        }
    }
}
```

```
follow();
```

4.2. Bucles i condicionals

El següent codi ens mostra algunes de les funcions predefinides més bàsiques, com poden ser les funcions que permeten desplaçar i girar el robot.

Inicialment trobem dues definicions de funcions `moveNoCrash()` i `setSpeedAndAcceleration(speed,acceleration)`, la primera definició té la intenció de simular la funció predefinida `exploreUltrasonic()` que fa que el robot es mogui vigilant de no xocar, fent ús de la funció pròpia del sensor ultrasònic `getDistance()`, la segona ens permet definir la velocitat i la acceleració dels motors situats al port A i B, fent ús de les funcions pròpies dels ports de motors `setSpeed(), setAcceleration()`.

Seguidament trobem dos bucles que fan fer diferents moviments al robot assignant a cada iteració una major o menor velocitat i acceleració, i una crida a la funció de `moveNoCrash()`.

```
def moveNoCrash(){
    forward();
    while(not($BESCAPE.isPressed())){
        if($S4.getDistance() < 20){
            stop();
            moveBack(10);
            while($S4.getDistance() < 25){
                rotateRight(15);
            }
```

```
            forward();
        }
    }
    stop();
}
def setSpeedAndAcceleration(speed, acceleration){
    $A.setSpeed(speed); $B.setSpeed(speed);
    $A.setAcceleration(acceleration); $B.setAcceleration(acceleration);
}
s = 100;
a = $A.getAcceleration()/8;
setSpeedAndAcceleration(s,a);
i = 0;
while(i < 4){
    moveFront(50);
    moveBack(50);
    s = s + 100;
    a = a*2;
    setSpeedAndAcceleration(s,a);
    i = i + 1;
}
i = 0;
while(i < 4){
    rotateLeft(360);
    rotateRight(360);
```

```
    s = s - 100;
    a = a/2;
    setSpeedAndAcceleration(s,a);
    i = i + 1;
}
moveNoCrash();
```

4.3. Recurrència i aritmètica

En aquest cas podem veure l'ús de l'aritmètica bàsica del llenguatge i la possibilitat de crear funcions recursives. La funció `recursiveFunc(speed,acc,iteration)`, fa que el robot comenci a desplaçar-se endavant a una certa velocitat i vagi incrementant-la a mesura que passa el temps.

```
def recursiveFunc(speed,acc,iteration){
    if(iteration <= 8) {
        forward();
        $A.setSpeed(speed);
        $B.setSpeed(speed);
        $A.setAcceleration(acc);
        $B.setAcceleration(acc);
        delay(500);
        recursiveFunc(speed*2,acc*2,iteration+1);
    }
}
recursiveFunc($A.getSpeed(),20,0);
stop();
```

```
a = 6 + 4;
b = 10 - a;
print(b);
b = 101;
b = b % 10;
print(b);
b = (b * (10 - 3))/2;
print(b);
waitForPress();
print("Press to continue and clear LCR");
clearDisplay();
color = $S2.getColor();
print("Color: ");
print(color);
waitForPress();
```

4.4. Importació de fitxers

En aquest apartat trobem dos fitxers, ja que mostrem la possibilitat de fer importació d'altres fitxers del tipus `rl`. El fitxer importat és vist com una llibreria de funcions, no s'hi pot crear un programa principal propi.

El següent codi és el fitxer que és importat, on tenim la definició de dues funcions `ultrasonicDistance()` i `goMaxDistance()`, la primera com ja indica el seu nom ens retorna la distància que està llegint en el moment de la crida el sensor ultrasònic, la segona com indica el comentari fa que el robot es dirigeixi a la distancia més gran de les quatre que se li passen per paràmetre, contant que la primera és endavant, la segona cap a la dreta, la

tercera cap endarrere i la quarta cap a l'esquerre.

```
def ultrasonicDistance() {
    return $S4.getDistance();
}
//El robot es dirigeix cap a on tingi més recorregut.
// dist0(0 graus), dist1 (90 graus), dist2 (180) ...
def goMaxDistance(dist0, dist1, dist2, dist3) {
    if(dist0 >= dist1 and dist0 >= dist2 and dist0 >= dist3) {moveFront(dist0 - 10
    elif(dist1 >= dist2 and dist1 >= dist0 and dist0 >= dist3){
        rotateRight(90);
        moveFront(dist1 - 10);
    }
    elif(dist2 >= dist1 and dist2 >= dist0 and dist2 >= dist3) {
        rotateRight(180);
        moveFront(dist2 - 10);
    }
    else {
        rotateRight(270);
        moveFront(dist3 - 10);
    }
}
/* Comentari
   de
   varies
   línies
```

```
*/
```

Aquí podem veure com fem l'import del fitxer anterior i fem ús de les funcions que hi han definides.

```
import funcs;
distFront = ultrasonicDistance();
rotateRight(90);
distRight = ultrasonicDistance();
rotateRight(90);
distBack = ultrasonicDistance();
rotateRight(90);
distLeft = ultrasonicDistance();
rotateRight(90);
goMaxDistance(distFront, distRight, distBack, distLeft);
```

5. Possibles extensions no implementades

5.1. Robolang com a llenguatge completament interpretat

La primera aproximació per implementar Robolang que es va provar va ser la d'un intèrpret. La idea era pujar el codi `.rl` al robot i que allà fos interpretat dinàmicament. Es va començar a explorar aquesta via per la seva similitud (en forma d'intèrpret) a ASL, a més de la netedat i claredat del codi generat.

Tot i així, van sorgir dos problemes principals que van acabar decantant el projecte cap al transpilador a Java que és actualment, ja que en realitat resultava ser una alternativa molt tediosa i poc eficient.

Primerament, el fet que la JVM que corre leJOS és antiga (Java 6) i incompatible amb ANTLR. Això ens va fer optar per generar un arbre i serialitzar-lo en un fitxer de text `.rltree`, que després seria interpretat pel robot.

Segonament, els recursos de hardware del robot són molt limitats, per tant, el rendiment de les proves que vam fer va ser força pobre, perquè a més d'executar el codi en si, havia d'interpretar (incloses tasques costoses com l'inferència de tipus) cada vegada.

5.2. MRs i arrays

Una possible funcionalitat que s'havia plantejat era la de crear un nou tipus de dades per tal de facilitar la feina al usuari a l'hora de fer moure el robot.

Bàsicament consistia en quatre tipus d'instruccions: `mf-rotacions` (move front), `mb-rotacions` (move back), `rh-angle` (rotar en sentit horari) i

`ra-angle` (rotar en sentit antihorari). Es limitava l'ús d'aquest tipus de dades en arrays per tal que l'usuari pogués fer un bucle `for` recorrent un seguit de moviments definits en aquest.

Es va implementar l'esmentat tipus de dades i també el d'arrays. El problema va ser que leJOS, al treballar amb Java 6, no admetia el mètode `Arrays.asList()` que permetia gestionar els arrays. Es va intentar buscar alguna alternativa factible sense èxit. Es va decidir suprimir aquesta opció del llenguatge, doncs, per una complicació excessiva en la implementació que tenia un impacte crític en la complexitat i el rendiment de la mateixa.

5.3. Reconeixement de veu

També hi va haver la intenció de implementar una funció que, mitjançant paraules clau (`GO, STOP, TURN LEFT, TURN RIGHT`), permetés controlar (moure) el robot.

No es va poder dur a terme degut a que el sensor de so necessari no venia inclòs en el paquet de compra del que disposava la universitat del model NXT 2.0. Tot i així la idea era aïllar el sensor a una certa alçada del robot per tal de que els motors no distorsionessin els valors, degut a que fan molt de soroll, i a partir d'aquí captar el so en 3 intervals de temps i analitzar els decibels en cada un, per així identificar de quina paraula es tractava.