

# LEGO roboten programazioa: NQC

(3.03 bertsioa, 1999/10/02)

Mark Overmars

Informatikako Departamentua  
Utrecht University  
P.O. Box 80.089, 3508 TB Utrecht  
the Netherlands

## Hitzaurrea

Lego MindStorms eta CyberMaster jostailu berri zoragarriak dira. Mota ezberdinetako robotak muntatzeko aukera ematen dute, eta gainera, konplexutasun handiko atazak bete ditzaten programa daitezke. Zoritxarrez, robotekin datorren softwarearen inguru grafikoa polita izan arren, eskaintzen dituen funtzioak oso mugatuak dira. Horregatik, bakarrik ataza errazak betetzeko erabil daiteke. Robotek dituzten aukera guztiez baliatzeko bestelako programazio inguru bat behar da: Dave Baum-ek NQC programazio hizkuntza garatu du, bereziki Lego robotekin erabiltzeko. Ez baduzu inoiz programarik idatzi, zaude lasai. NQC oso erraza da erabiltzeko, eta gidaliburu honek behar duzun guztia argituko dizu. Berez, NQC-z robotak programatzea ordenagailu arrunt bat programatzea baino errazagoa da, eta era erraz batean programatzaile bilakatzeko aukera emango dizu.

Programak era erraz batean editatzeko RCX Command Center dugu. Utilitate honek programak idazten, robotera transferitzen, eta robota martxan jartzen eta gelditzen lagunduko dizu. RCX Command Center-en funtzionamendua testu-prozesadore baten antzekoa da, baina gehigarri batzuekin. Gidaliburu honetan RCX Command Center erabiliko dugu (3.0 bertsioa edo ondorengoa). Ondorengo Web helbidean aurkituko duzu:

<http://www.cs.uu.nl/people/markov/lego/>

RCX Command Center Windows sistema eragilea ('95, '98, 2000, NT) duten PCtan funtzionatzen duen utilitatea da. RCX Command Center erabili aurretik derrigorrezkoa da Legoren softwarea gutxienez behin egikaritzea. Legoren softwareak RCX Command Center-ek behar dituen osagai batzuk instalatzen ditu. NQC programazio hizkuntza beste plataforma batzuetan erabil daiteke. Interneteko ondoko helbidean aurki dezakezu:

<http://www.enteract.com/~dbaum/lego/nqc/>

Gidaliburu honetako gehiengoa beste plataformetan erabilgarria da (NQCko 2.0 edo geroko bertsioa erabiltzen baduzu), baina tresna batzuk eta kolore kodea galduko dituzu. Izenak ere ezberdinak izango dira, adibidez motoreenak.

Gidaliburu hau jarraitzeko MindStorms robota behar duzu. Eduki gehienak CyberMaster robotekin ere erabil daitezke, nahiz eta ezaugarri batzuk bestelako robot hauekin erabilgarri ez izan. Motoreen izenak ezberdinak izango dira, beraz, CyberMaster baduzu adibideetan aldaketa batzuk egin beharko dituzu.

## Eskerrak

Dave Baum-i eskerrak eman nahi dizkiot, NQC garatu duelako. Baita Kevin Said-ri ere, gidaliburu honetako lehen zatiaren lehen bertsioa idatzi zuelako.

# Aurkibidea

<b>Hitzaurrea</b>	<b>2</b>
Eskerrak	2
<b>Aurkibidea</b>	<b>3</b>
<b>I. Zure lehen programa</b>	<b>5</b>
Robot baten muntaketa	5
RCX Command Center abiarazi	5
Programaren edizioa	6
Programaren egikaritzea	7
Hutsegiteak programan	7
Abiaduraren aldaketak	8
Laburpena	8
<b>II. Interesgarriagoa den programa bat</b>	<b>9</b>
Biraketak	9
Instrukzioen errepikapena	9
Azalpenen sartzea	10
Laburpena	11
<b>III. Aldagaien erabilpena</b>	<b>12</b>
Mugimendua kiribilean	12
Zenbaki aleatorioak	13
Laburpena	13
<b>IV. Kontrol egiturak</b>	<b>14</b>
if instrukzioa (baldintzazkoa)	14
do instrukzioa	15
Laburpena	15
<b>V. Sentsoreak</b>	<b>16</b>
Sentsorearen zain	16
Oztopoak ekiditen	16
Argi-sentsoreak	17
Laburpena	18
<b>VI. Atazak eta subrutinak</b>	<b>19</b>
Atazak	19
Subrutinak	20
Funtzioak	20
Makroen definizioa	21
Laburpena	22
<b>VII. Soinuak eta musika</b>	<b>23</b>
Soinu aukera	23
Musika jotzen	23
Laburpena	24
<b>VIII. Motoreetan sakontzen</b>	<b>25</b>
Emeki geldituz	25
Agindu aurreratuak	25
Motorearen abiaduraren aldaketa	26
Laburpena	26
<b>IX. Sentsoreetan sakontzen</b>	<b>27</b>
Sentsore mota eta moduak	27
Errotazio-sentsorea	28
Nola jarri sentsore bat baino gehiago sarrera batean	29
Nola egin hurbiltasun-sentsore bat	30

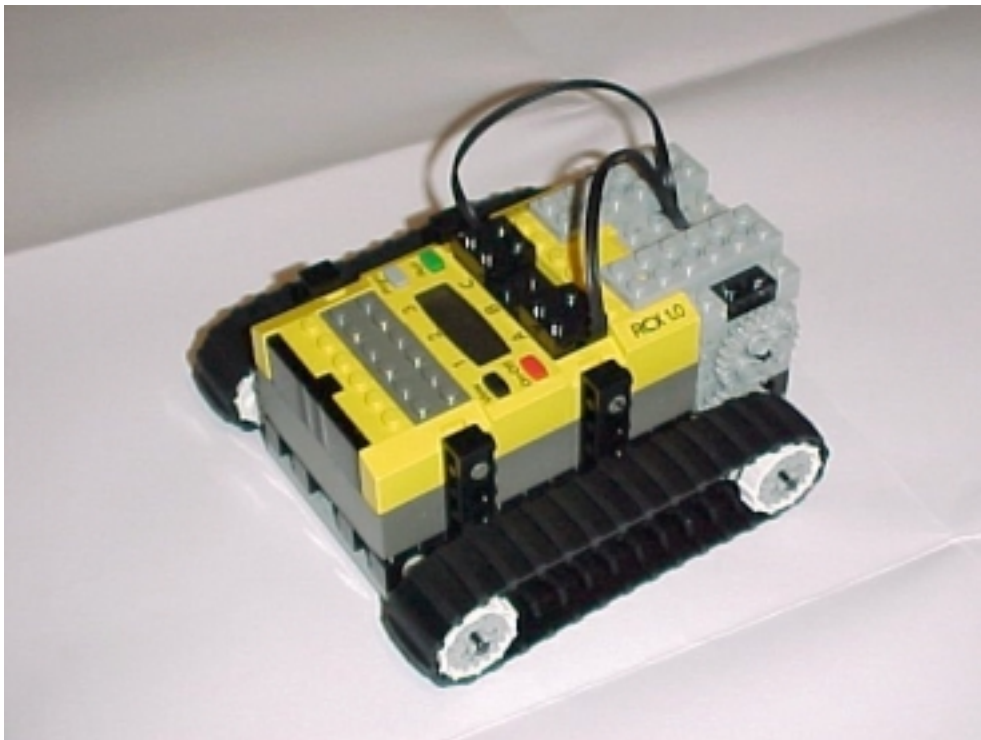
Laburpena _____	31
<b>X. Ataza paraleloak _____</b>	<b>32</b>
Programa akastun bat _____	32
Atazen gelditzea eta berrabiaraztea _____	32
Semaforoen erabilpena _____	33
Laburpena _____	34
<b>XI. Roboten arteko komunikazioak _____</b>	<b>35</b>
Aginduen igorpena _____	35
Nagusiaren izendapena _____	36
Kontuan hartzekoa _____	36
Laburpena _____	37
<b>XII. Agindu gehiago _____</b>	<b>38</b>
Tenporizadoreak _____	38
Pantaila _____	38
Datuen erregistroa _____	39
<b>XIII. NQC-ren erreferentzia azkarra _____</b>	<b>40</b>
Instrukzioak _____	40
Baldintzak _____	40
Espresioak _____	41
RCX funtzioak _____	41
RCX konstanteak _____	43
Erreserbatutako hitzak _____	43
<b>XIV. Azken oharrak _____</b>	<b>44</b>
<b>XV. Itzultzailearen oharra _____</b>	<b>45</b>

## I. Zure lehen programa

Kapitulu honetan programa erraz bat nola idatzi ikasiko duzu. Programa honekin, robotak lau segundoz aurrera egingo du, eta ondoren, beste lau segundoz atzera. Ez da oso ikusgarria, baina programazioaren oinarrietan sartzeko bide egokia da. Gainera, programatzea erraza dela erakutsiko dizu. Baina programa idatzi aurretik robot bat beharko dugu.

### Robot baten muntaketa

Eskuliburu honetan erabiliko dugun robota “Konstruktopedia”-ren 39-46<sup>1</sup> orrialdetan deskribatzen den top-secret robotaren bertsio erraz bat da. Oinarrizko xasisa baino ez dugu erabiliko. Ken izeaiozu aurreko aldean duen guztia, hau da, besoak eta ukitze-sentsoreak. Kableak ere ez ditugu modu berdinean konektatuko: RCX-n kanpo aldera konektatu behar dira irudian ikusten den moduan. Hau oso garrantzitsua da robotak behar duen noranzkoan mugitu dadin. Robotak irudikoaren itxura eduki behar du:



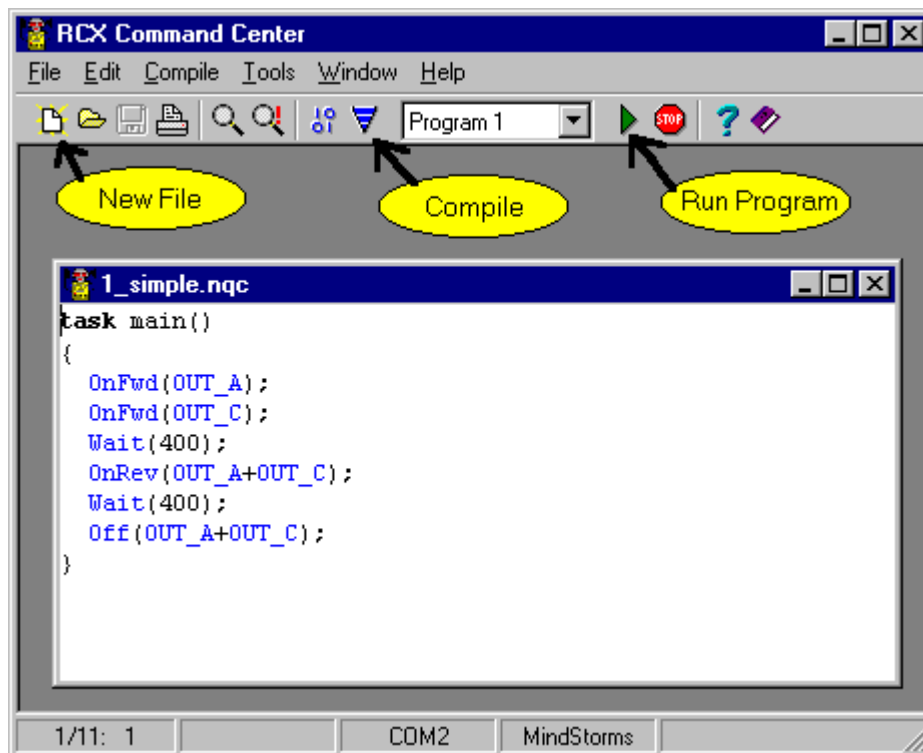
Bestetik, egiazta ezazu infragorrien dorrea ordenagailuari behar bezala konektatuta dagoela, eta irismen luzerako konfiguraturua dagoela (egiaztapenak egiteko RIS softwarea erabil dezakezu).

### RCX Command Center abiarazi

Programak idazteko RCX Command Center erabiliko dugu. Abiaraz ezazu RcxCC ikonoaren gainean klik-bikoitza eginez (ordenagailuan instalatuta duzula suposatu dut, hala ez bada, jaitsi ezazu Internet-etik<sup>2</sup>, deskonprimitu ezazu eta jar ezazu gustukoen duzun direktorioan). Programak robota non aurki dezakeen galdetuko dizu. Piztu ezazu RCX eta saka ezazu **OK**. Programak robota automatikoki aurkituko du (segur aski). Orain, erabiltzailearen interfazea ondorengo irudian ikusten den eran agertuko da (leihorik gabe).

<sup>1</sup> Eskuliburu honetan aipatzen den “Konstruktopedia” RIS (Robotics Invention System) 1.0-ri dagokiona da (9719 erref.). Bertsio hau ez zen Euskal Herriean komertzializatu (itzultzailearen oharra).

<sup>2</sup> <http://www.cs.uu.nl/people/markov/lego/>



Editoreak testu-editore arrunt baten itxura du, ohiko menuarekin, eta fitxategiak irekitzeko, gordetzeko, inprimatzeko... botoiekin. Baina programak konpilatzeko, robotera transferitzeko eta robotetik informazioa jasotzeko botoi berezi batzuk ere eskaintzen ditu, oraingoz, alde batera utziko ditugunak.

Idatz dezagun programa berri bat. Horretarako, saka ezazu **New File** botoia dokumentu berri bat sortzeko.

## Programaren edizioa

Orain idatz ezazu ondorengo programa:

```

task main()
{
    OnFwd(OUT_A);
    OnFwd(OUT_C);
    Wait(400);
    OnRev(OUT_A+OUT_C);
    Wait(400);
    Off(OUT_A+OUT_C);
}

```

Hasieran korapilatsua dirudi, eta horregatik, aztertuko dugu. NQC-n programak atazaz osaturik daude. Gure programak ataza bat du, *main* (nagusia) izenekoa. Programa guztiek *main* izeneko ataza bat eduki behar dute, hain zuzen, robotak hori egikaritzen baitu. VI kapituluan gauza gehiago ikasiko duzu atazaz. Ataza bat agindu multzo batez osaturik dago, instrukzio izenaz ere ezagutzen direnak. Bi giltzek instrukzio multzoa mugatzen dute, honela, ez da dudarik egongo zeintzuk diren atazari dagozkion instrukzioak. Instrukzio bakoitzaren bukaeran puntu eta koma karakterea jarri behar da. Honela, argi gelditzen da non dauden instrukzio baten hasiera eta bukaera. Orokorrean, ataza baten itxura ondoko hau izango da:

```

task main()
{
    1instrukzioa;
    2instrukzioa;
    ...
}

```

Gure programak sei instrukzio ditu. Ikus ditzagun banan-banan:

```
OnFwd(OUT_A);
```

Instrukzio honek A irteera hasieratzeko eskatzen dio: RCX-n A izena duen irteeran konektatutako motoreak aurrera egingo du. Ahal duen azkarren mugituko da, aurrez bestelako abiadura finkatzen ez den artean. Abiadura nola aldatu aurrerago ikasiko duzu.

```
OnFwd(OUT_C);
```

Instrukzio bera da, baina oraingo honetan C motorea jarriko dugu martxan. Bi instrukzio hauen ondoren bi motoreak martxan egongo dira, eta robota aurrerantz mugituko da.

```
Wait(400);
```

Orain denbora tarte batez itxaroteko garaia da. Instrukzio honek programaren egikaritzea 4 segundoz geldituko du. Argumentua, hau da, parentesi artekoa, "tik" kopurua adierazten du ("tik" bat segundo baten ehunen bat da). Beraz, zenbat denboraz itxaron behar den oso era zehatzean eman daiteke.

Programak lau segundoz ez du ezer egingo, beraz, robotak aurrera jarraituko du.

```
OnRev(OUT_A+OUT_C);
```

Robota behar beste urrutiratu ondoren, itzultzeko garaia da. Instrukzio honek motoreen noranzkoa aldatuko du. Ikus dezakezunez, instrukzio bakar batez egin dugu, `OUT_A+OUT_C` erabiliz. Programaren hasieran ere, modu honetan egin genezakeen instrukzio bakar bat erabiliz.

```
Wait(400);
```

Berriz lau segundoz itxarongo dugu.

```
Off(OUT_A+OUT_C);
```

Eta bukatzeko bi motoreak geldituko ditugu.

Hau da programa osoa. Motoreak 4 segundoz aurrera egingo dute, beste lau segundoz atzera, eta bukatzeko geldituko dira.

Beharbada ohartu zara editoreak kolore kode bat erabiltzen duela. Kolore hauek automatikoki agertzen dira. Urdinez agertzen den gutzia robotari luzatzen zaion agindua da. **Task** hitza letra lodiz agertzen da NQC-n garrantzizko hitz bat delako (erreserbatua). Aurrerago ikusiko dugun eran, gainerako hitz erreserbatuak ere lodiz ageriko dira. Koloreak oso baliagarriak dira edizioan hutsegiteak ekiditeko.

## Programaren egikaritzea

Programa idatzi ondoren konpilatu behar da (hau da, robotak ulertu eta egikari dezakeen kodea bihurtu) eta infragorrien atakaren bitartez robotera transferitu behar da (download programan). Bi betebeharrak botoi bakar bat zapalduz egin daiteke (ikus aurreko irudian). Saka ezazu botoia, eta programaren edizioan hutsegiterik egin ez baduzu, konpilatu eta robotera transferituko da (programan hutsegiterik badago, aplikazioak jakinaraziko du; ikus hurrengo atala).

Orain programa egikari dezakezu. Saka ezazu zure roboteko botoi berdea (Run), edo, saka ezazu editorean agertzen den Run botoia. Espero zenuena egin al du robotak? Hala ez bada, seguruena kableak behar bezala konektatuta ez daudelako izango da.

## Hutsegiteak programan

Programa bat idazten denean ez da harrigarria izaten errore bat edo beste egitea. Konpiladoreak egindako hutsegiteen berri ematen digu leihoaren beheko aldean ematen dituen mezuen bitartez (ikus hurrengo irudia).

```
1_errors.nqc
task main()
{
  OnFwd(OUT_D);
  OnFwd(OUT_C);
  Wait(400);
  OnRev(OUT_A+OUT_C);
  Wait(400);
  Off(OUT_A+OUT_C);
}

line 3: Error: undefined variable 'OUT_D'
```

Automatikoki lehen errorea aukeratzen du (guk motorearen izena gaizki idatzi dugu). Errore bat baino gehiago dagoenean errore-mezuen gainean klik egin dezakezu bat edo beste aukeratzeko. Maiz, programaren hasieran errore bat gertatzen denean, beste batzuk sortzen ditu errenkan, beraz, hobe da hasierako errorea zuzentzea eta berriz programa konpilatzea. Koloreen kodeak errore asko ekiditen ditu, adibidez, azken lerroan `Off` idatzi dugu `Off` -en orde. Ezagutzen ez duen agindu bat denez ez da urdinez agertzen.

Badaude konpiladoreak detektatzen ez dituen erroreak ere. `OUT_B` idazten badugu, konpiladoreak hutsegite honen berririk ez du emango motore hori existitzen delako (nahiz eta gure robotean ez erabili). Beraz, robotak ez badu espero dugun portaera erakusten, litekeena da programan zerbait gaizki egotea.

## Abiaduraren aldaketak

Ohartuko zinen robota azkar dabilela. Besterik agindu ezean, ahal duen azkarren mugituko da. Abiadura aldatzeko `SetPower()` instrukzioa erabiliko dugu. Potentzia 0 eta 7 bitarteko zenbaki bat: 0 motelena (baina gelditu gabe) eta 7 azkarrena. Jarraian, gure programaren bertsio berri bat aurkezten da, robota polikiago mugi dadin.

```
task main()
{
  SetPower(OUT_A+OUT_C,2);
  OnFwd(OUT_A+OUT_C);
  Wait(400);
  OnRev(OUT_A+OUT_C);
  Wait(400);
  Off(OUT_A+OUT_C);
}
```

## Laburpena

Kapitulu honetan NQC-z zure lehen programa idatzi duzu, eta horretarako RCX Comand Center erabili duzu. Orain badakizu nola idatzi programa bat, nola transferitu robotera eta nola egikaritu robotean. RCX Command Center-ekin gauza gehiago egin daitezke, eta zeintzuk diren jakin nahi baduzu, irakur ezazu bere dokumentazioa. Eskuliburu honen helburua NQC programazio hizkuntzan sakontzea da, eta, RCX Command Center-en ezaugarri batzuk baino ez dira aipatzen.

NQC-ri buruz gauza batzuk ikasi dituzu. Lehena, programa guztiek `main` izeneko ataza bat (task) dutela, eta hau izango dela robotak egikaritutako duena. Motoreak erabiltzeko instrukzio garrantzitsuenak ere ikasi dituzu: `OnFwd()`, `OnRev()`, `SetPower()` y `Off()`. Hauekin batera `Wait()` instrukzioa nola erabiltzen den ikasi duzu.

## II. Interesgarriagoa den programa bat

Gure lehen programa ez zen oso ikusgarria. Orain interesgarriagoa den beste bat egiten saiatuko gara. Programa urratsez urrats egingo dugu, NQC programazio hizkuntzaren garrantziko ezaugarriak sartuz.

### Biraketak

Robotak biraketak egiteko modu ezberdinak daude. Oinarritzkoenak motore bat gelditzea edo bere biraketa noranzkoa aldatzea dira. Hona hemen adibide bat. Idatz ezazu, eta ondoren, transferi ezazu robotera eta egikari ezazu. Pixka bat aurrera egin ondoren, eskuin aldera 90°ko egingo du.

```
task main()
{
    OnFwd(OUT_A+OUT_C);
    Wait(100);
    OnRev(OUT_C);
    Wait(85);
    Off(OUT_A+OUT_C);
}
```

Bigarren `Wait()` instrukzioan 85 ez diren balioak probatu ditzakezu 90°-ko biraketa zehatzagoa izan dadin. Hau robota mugitzen den lur motaren arabera da. Aldaketa hau errazteko, zenbaki hau ordezkatu duen izen bat erabil dezakegu. NQC-n balio konstanteak defini daitezke, ondoko programan erakusten den moduan.

```
#define AURRERA_DENBORA 100
#define BIRAKETA_DENBORA 85

task main()
{
    OnFwd(OUT_A+OUT_C);
    Wait(AURRERA_DENBORA);
    OnRev(OUT_C);
    Wait(BIRAKETA_DENBORA);
    Off(OUT_A+OUT_C);
}
```

Lehenengo bi lerroetan bi konstante definitzen dira. Hauek, programan nahi den tokietan erabil daitezke. Konstanteak definitzea ideia ona da bi arrazoiengatik: programa irakurgarriagoa izango da eta balioak aldatzea errazagoa izango da. RCX Command Center-ek kolore propioa ematen die konstanteen definizioei. VI kapituluaren ikusiko dugun moduan, konstanteak ez diren beste elementu batzuk defini ditzakezu.

### Instrukzioen errepikapena

Orain, robotak ibilbide karratu bat deskribatzeko behar den programa idazten saiatuko gara. Honek, zera esan nahi du: aurrera egiten du, 90°ko biraketa, aurrera egiten du, 90°ko biraketa... Aurreko programako kode zatia lau aldiz errepika dezakegu, baina errazagoa da `repeat` instrukzioa erabiliz.

```
#define AURRERA_DENBORA 100
#define BIRAKETA_DENBORA 85

task main()
{
    repeat(4)
    {
        OnFwd(OUT_A+OUT_C);
        Wait(AURRERA_DENBORA);
        OnRev(OUT_C);
        Wait(BIRAKETA_DENBORA);
    }
    Off(OUT_A+OUT_C);
}
```

**repeat** instrukzioaren ondoren parentesi artean dagoen zenbakia errepikapen kopurua adierazten du. Errepikatuko diren instrukzioak giltzen artean idazten dira, atazetan egiten den modu berean. Beharbada ohartuko zara aurreko programan instrukzioak koskatu ditugula. Hau ez da beharrezkoa, baina koskatzearen ondorioz programa era errazagoan irakurtzen da.

Instrukzio honekin bukatzeko, robotari ibilbide karratuko hamar itzuli emateko eskatuko diogu. Hona hemen programa:

```
#define AURRERA_DENBORA 100
#define BIRAKETA_DENBORA 85

task main()
{
    repeat(10)
    {
        repeat(4)
        {
            OnFwd(OUT_A+OUT_C);
            Wait(AURRERA_DENBORA);
            OnRev(OUT_C);
            Wait(BIRAKETA_DENBORA);
        }
    }
    Off(OUT_A+OUT_C);
}
```

Orain, errepikapenak egiteko instrukzio bat beste baten barruan jarri dugu. Honi errepikapen instrukzio kabiatura deritzo. Nahi dituzun errepikapen instrukzio guztiak kabiaturik ditzakezu. Begira iezaiezu programa honetan erabilitako giltzei eta koskatzeei: ataza lehen giltzan hasi eta azkenekoan bukatzen da. Lehen errepikapena bigarren giltzan hasten da eta bosgarrenean bukatzen. Bigarren errepikapena, kabiatura, hirugarren giltzan hasten da eta laugarrenean bukatzen. Ikus dezakezun moduan giltza kopurua beti bikoitia izango da, eta haien arteko kode zatiak koskatu egiten dira.

## Azalpenen sartzea

Zure programa irakurgarriagoa izan dadin, komeni da azalpenak tartekatzea. Lerro batean // jartzen duzun bakoitzean, konpiladoreak hortik aurrerakoa ez du kontuan hartuko, beraz nahi duzuna idatz dezakezu bertan. Azalpen luze bat /\* eta \*/ artean idatz dezakezu (lerro batzuetan zehar idazteko aukera ematen du). Azalpenak kolore berdez agertuko dira RCX Command Center-en. Programa osoa ondoko hau izango da:

```

/* 10 karratu

   Mark Overmars-ek egina

Programa honi esker robotak 10 itzuli emango ditu karratu
baten gaineratik
*/

#define AURRERA_DENBORA 100 // Aurrera egiteko denbora
#define BIRAKETA_DENBORA 85 // 90ºz biratzeko denbora

task main()
{
  repeat(10) // 10 itzuli egin
  {
    repeat(4)
    {
      OnFwd(OUT_A+OUT_C);
      Wait(AURRERA_DENBORA);
      OnRev(OUT_C);
      Wait(BIRAKETA_DENBORA);
    }
  }
  Off(OUT_A+OUT_C); // Orain motoreak geldituko dira
}

```

## Laburpena

Kapitulu honetan azalpenak tartekatzen eta **repeat** instrukzioa erabiltzen ikasi duzu. Baita, instrukzioak nola kabiatu eta koskatzea nola erabili ere. Honek guztiak edozein motako ibilbidea jarraituko duen robot bat programatzeko aukera ematen dizu. Aurrera jarraitu aurretik, komeni zaizu kapitulu honetako programetan aldaketa batzuk egitea.

### III. Aldagaien erabilpena

Aldagaien erabilpena edozein programazio hizkuntzaren alderdi garrantzitsua da. Aldagaiak datuen balioak gordetzeko aukera ematen duten memoria posizioak dira. Balio hauek programaren toki ezberdinetan erabili ahal izango ditugu, eta hala nahi badugu, beren balioak aldatzeko aukera izango dugu. Aldagaien erabilpena adibide baten bitartez deskribatuko dut.

#### Mugimendua kiribilean

Demagun aurreko kapituluko programa moldatu nahi dugula robota kiribil bat deskribatzen mugi dadin. Hau, zain egon behar duen denbora pixkanaka-pixkanaka luzatuz lor daiteke, aurrerako mugimenduak gero eta luzeagoak izan daitezten. Hau da, aldi bakoitzean AURRERA\_DENBORA-ren balioa luzatuko dugu. Baina, nola egin dezakegu hori? AURRERA\_DENBORA konstante bat da, eta konstanteen balioak ezin dira aldatu. Bere ordean, aldagai bat beharko dugu. Oso erraza da NQC-n aldagaiak definitzea. 32 erabil ditzakezu, eta bakoitzari nahi duzun izena eman diezaiokezu. Hona hemen Kiribila programa

```
#define BIRAKETA_DENBORA 85

int aurrera_denbora;           // aldagai bat definitu

task main()
{
    aurrera_denbora = 20;      // hasierako balioa esleitu
    repeat(50)
    {
        OnFwd(OUT_A+OUT_C);
        Wait(aurrera_denbora); // zain egoteko aldagaia erabili
        OnRev(OUT_C);
        Wait(BIRAKETA_DENBORA);
        aurrera_denbora += 5;  // aldagaiaren balioa handitu
    }
    Off(OUT_A+OUT_C);
}
```

Lerro aipagarrietan azalpenak ematen dira. Hasieran aldagai bat hasieratu dugu, **int** erreserbatutako hitzaren ondoren aukeratutako izena idatziz (nahiz eta derrigorrezkoa ez izan, aldagaien izenetan letra xeheak erabiliko ditugu, eta aldagaien izenetan letra larriak). Izenaren hasiera letra bat izango da, baina hortik aurrera digituak eta beheko gidoia ere erabil daitezke. Ezin da beste inolako sinbolorik erabili (arau hau ataza eta konstanteen izenetan ere bete behar da). **int** hitza zenbaki osoentzat erreserbatzen da. Mota honetako aldagaietan zenbaki osoak baino ezin dira gorde. Bigarren lerro aipagarrian 20 balioa esleitu diogu aldagaiari. Hortik aurrera aldagaia erabiltzen dugun bakoitzean bere balioa 20 izango da. Ondoren, zain egoteko denbora ordezkatzeko duen aldagaia erabiltzen duen errepikapen begizta dator. Begizta honen bukaeran aldagaiaren balioari bost unitate gehitzen zaizkio. Honela, lehen aldian robotak 20 “tik” pasa arte aurrera egiten du, bigarrean 25, hirugarrean 30...

Aldagai bati balioak gehitzeaz gain, bestelako eragiketa aritmetikoak egin ditzakegu ere: biderketak \*= erabiliz, kenketak -= erabiliz eta zatiketak /= erabiliz (zatiketaren kasuan emaitza balio oso gertuenera borobiltzen da). Gainera, aldagai bat beste bati gehi diezaiokezu, edo espresio konplexuagoak idatzi. Ikus itzazu ondoko adibideak:

```
int aaa;
int bbb, ccc;

task main()
{
    aaa = 10;
    bbb = 20 * 5;
    ccc = bbb;
    ccc /= aaa;
    ccc -= 5;
    aaa = 10 * (ccc + 3); // orain aaa-ren balioa 80 izango da
}
```

Begira iezaiozu bigarren lerroari, bertan aldagai batzuk hasieratzen baitira. Nahi izanez gero, hiru aldagaiak lerro bakar batean jar daitezke.

## Zenbaki aleatorioak

Aurreko programetan robotak egin behar zuena modu zehatzean definitu dugu. Baina interesak gora egiten du robotak zer egingo duen ezagutzen ez dugunean. Bere mugimenduak neurri batean aleatorioak nahi ditugu. NQC-n zenbaki aleatorioak erabil ditzakegu. Hurrengo programak erabiltzen ditu era aleatorio batean mugitzeko. Zenbaki aleatorio batek zehaztutako denboraz aurrera egin ondoren, era aleatorio batean biratuko da.

```
int aurrera_denbora, biraketa_denbora;

task main()
{
    while(true)
    {
        aurrera_denbora = Random(60);
        biraketa_denbora = Random(40);
        OnFwd(OUT_A+OUT_C);
        Wait(aurrera_denbora);
        OnRev(OUT_A);
        Wait(biraketa_denbora);
    }
}
```

Programak bi aldagai definitu ondoren balio aleatorioak esleitzen dizkie. `Random(60)` 0 eta 60 arteko zenbaki aleatorio bat da (0 edo 60 ere izan daiteke). Begizta bakoitzean zenbakiak ezberdinak izango dira. `Wait(Random(60))` idatziko bagenu, ez guke aldagaiak erabili beharrik izango.

Hemen begizta mota berri bat ikusi dugu. Errepikapen instrukzioa erabili ordez, `while(true)` idatzi dugu. `while` (bitartean) instrukzioak jarraian dituen instrukzioak errepikatzen ditu parentesi artean dagoen baldintza egiazkoa den bitartean. `True` (egiazkoa) erreserbatutako hitza beti betetzen da, beraz, giltzen artean dauden instrukzioak etengabe errepikatuko dira nahi dugun arte. IV kapituluari gehiago ikasi ahal izango duzu `while` kontrol egituraz.

## Laburpena

Kapitulu honetan aldagaiak nola erabili ikasi duzu. Aldagaiak oso erabilgarriak dira, baina roboten murrizketen ondorioz pixka bat mugatuak dira. Gehienez 32 aldagai defini ditzakezu<sup>3</sup>, eta zenbaki osoak gordetzeko baino ezin dituzu erabili. Baina roboten atazen gehienetan hau nahikoa da.

Bestetik, zenbaki aleatorioak nola sortu ikasi duzu, robotari ezusteko portaera emateko. Amaitzeko `while` instrukzioa amaigabeko begiztak egiteko nola erabili ikusi dugu.

---

<sup>3</sup> Itzultzailearen oharra: RCX 2.0 firmwarea eta NQC-ren azken bertsioa erabiliz aldagai gehiago erabil daitezke.

## IV. Kontrol egiturak

Aurreko kapituluetan **repeat** eta **while** instrukzioak ikusi ditugu. Kontrol instrukzio hauek programako beste hainbat instrukzioen egikaritzea dakarte. “Kontrol egiturak” izenekoak dira. Kapitulu honetan beste kontrol egiturak ikusiko ditugu.

### if instrukzioa (baldintzazkoa)

Batzuetan, programaren zati bat hainbat egoera zehatzetan bakarrik egikaritzea nahi dugu. Kasu horietan **if** instrukzioa erabiltzen da. Ikus dezagun adibide baten bitartez. Aurreko kapituluetan erabilitako programari aldaketa berri bat egingo diogu. Robotak zuzen aurrera egin ondoren ezkerrera ala eskuinera biratzea nahi dugu. Hau egiteko, berriro beharko ditugu zenbaki aleatorioak. 0 edo 1 balioa eduki dezaken zenbaki aleatorio bat erabiliko dugu. Zenbakiaren balioa 0 bada, eskuinera biratuko da, eta 1 denean berriz, ezkerrera. Programa hau da:

```
#define AURRERA_DENBORA 100
#define BIRAKETA_DNBORA 85

task main()
{
    while(true)
    {
        OnFwd(OUT_A+OUT_C);
        Wait(AURRERA_DENBORA);
        if (Random(1) == 0)
        {
            OnRev(OUT_C);
        }
        else
        {
            OnRev(OUT_A);
        }
        Wait(BIRAKETA_DNBORA);
    }
}
```

**if** instrukzioa nolabaiteko antza du **while** instrukzioarekin. Parentesi artean dagoen baldintza betetzen bada, giltzen artean dagoen kodea egikarituko da. Betetzen ez bada, berriz, **else**-ren ondoren giltzen artean dagoen kodea egikarituko da. Azter dezagun erabili dugun baldintza: `Random(1) == 0`. Honek zera esan nahi du: `Random(1)` 0 denean baldintza egiazkoa izango da. Beharbada, ohartu zara programan `=`-ren ordez `==` erabili dugula. Hau, konparaketak eta balioen esleipenak bereizteko egiten da.

Balioak era ezberdinetan konpara daitezke. Ondoko hauek dira garrantzitsuenak:

<code>==</code>	berdin
<code>&lt;</code>	txikiago baino
<code>&lt;=</code>	txikiago edo berdin
<code>&gt;</code>	handiago baino
<code>&gt;=</code>	handiago edo berdin
<code>!=</code>	ezberdin

Baldintzak konbina daitezke: `&&`-k “eta” esan nahi du eta `||` “edo”. Ikus ditzagun adibide batzuk:

<code>true</code>	beti egiazkoa
<code>false</code>	inoiz ere ez egiazkoa
<code>ttt != 3</code>	egiazkoa ttt-k 3 balioa ez duenean
<code>(ttt &gt;= 5) &amp;&amp; (ttt &lt;= 10)</code>	egiazkoa ttt 5 eta 10 arteko balio bat denean
<code>(aaa == 10)    (bbb == 10)</code>	egiazkoa aaa edo bbb-ren (edo biena) balioa 10 denean.

**if** instrukzioak bi zati ditu: baldintza eta berehala datorren zatia baldintza egiazkoa denean beteko da, eta **else**-ren ondorengo baldintza egiazkoa ez denean. **else** instrukzioa eta ondorengo aukerakoak dira, beraz, baldintza faltsua denean ez badu ezer egin behar, ez dugu besterik jarriko.

## do instrukzioa

Hona hemen beste kontrol egitura bat, hain zuzen, **do** instrukzioa. Honelako sintaxia du:

```
do
{
    instrukzioak;
}
while (baldintza);
```

**do**-ren ondoko giltzen artean dauden instrukzioak baldintza egiazkoa den bitartean egikarituko dira. Baldintzaren egitura **if** instrukzioaren kasuan deskribatu den modukoa da. Jarraian adibide bat eskaintzen da. Robota 20 segundoz era aleatorioan mugitu ondoren geldituko da.

```
int aurrera_denbora, biraketa_denbora, denbora_guztira;

task main()
{
    denbora_guztira = 0;
    do
    {
        aurrera_denbora = Random(100);
        biraketa_denbora = Random(100);
        OnFwd(OUT_A+OUT_C);
        Wait(aurrera_denbora);
        OnRev(OUT_C);
        Wait(biraketa_denbora);
        denbora_guztira += aurrera_denbora; denbora_guztira += biraketa_denbora;
    }
    while (denbora_guztira < 2000);
    Off(OUT_A+OUT_C);
}
```

Adibide honetan bi instrukzio lerro berean jarri ditugu. Hau egiteko ez dago inolako oztoporik; lerro batean nahi dituzun instrukzio guztiak jar ditzakezu (beti ere, tartean puntu eta komak jartzen badituzu). Baina programa errazago irakurtzeko ez da gomendagarria.

**do** instrukzioaren portaera eta **while**-rena oso antzekoak dira, baina, **while** instrukzioan baldintza hasieran egiaztatzen den bitartean, **do** instrukzioan amaieran egiaztatzen da. **while** kontrol egitura erabiltzean, gerta daiteke dagozkion instrukzioak ez egikaritzea, baina **do** egiturakoak gutxienez behin egikarituko dira.

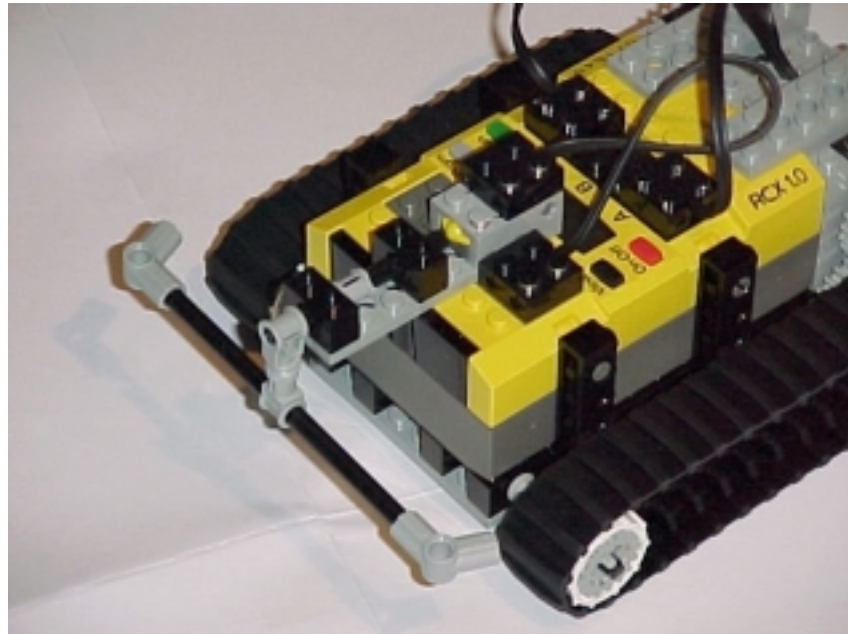
## Laburpena

Kapitulu honetan bi kontrol egitura berri ikusi ditugu: **if** eta **do** instrukzioak. **repeat** eta **while**-rekin batera programaren nondik norakoa kontrolatzen dute. Garrantzi handikoa da beren funtzionamendua ondo ulertzea. Horregatik, aurrera jarraitu aurretik zure kontura adibide gehiago ebaztea komeni zaizu.

Ikusi dugun beste gauza bat, lerro batean instrukzio bat baino gehiago idatz daitekeela da.

## V. Sentsoreak

LEGo roboten ezaugarririk interesgarrienetako bat sentsoreak konektatzeko duen aukera da, hortik aurrera sentsoreen irakurketetan oinarrituta erabakiak hartzeko. Hasteko, sentsore bat jarriko diogu robotari. Horretarako, “Construktopedia”-ren 28. orrialdeko laugarren irudian aurkezten den muntaketa egin behar duzu. Aldaketaren bat egin dezakezu, irudiko robotean ikusten den moduan:



Konekta ezazu sentsorea RCX-ko 1 sarreran.

### Sentsorearen zain

Egingo dugun lehen programarekin robotak aurrera egingo du zerbaiten aurka talka egin arte. Ikus dezagun:

```
task main()
{
  SetSensor(SENSOR_1, SENSOR_TOUCH);
  OnFwd(OUT_A+OUT_C);
  until (SENSOR_1 == 1);
  Off(OUT_A+OUT_C);
}
```

Hemen bi lerro aipagarri dago: programaren lehenengo lerroan robotari nolako sentsorea erabili behar dugun jakinarazi diogu. `SENSOR_1` sentsorea konektatua dagoen sarrera da. Beste bi sarreraren izenak `SENSOR_2` eta `SENSOR_3` dira. `SENSOR_TOUCH` ukitze-sentsore bat dela adierazten du. Argi-sentsore bat erabiltzeko `SENSOR_LIGHT` idatzi beharko genuke. Sentsorea konfiguratu ondoren, programak bi motoreak martxan jartzen ditu eta motorea aurrerako bidea hartzen du. Ondorengo instrukzioa oso erabilgarria da. Baldintza honek `SENSOR_1` sentsorearen balioa 1 izan behar duela dio, hau da, sentsorea sakatuta egon behar duela. Sentsorea sakatuta ez dagoen artean, balioa 0 izango da. Beraz, instrukzio honi esker programa zain gelditzen da sentsorea presionatu arte. Une horretan motoreak geldituko dira eta ataza amaituko da.

### Oztopoak ekiditen

Saia gaitezen orain oztopoak ekiditen dituen robot bat egiten. Robotak talka egiten duen bakoitzean atzera egingo du, eta biraketa txiki bat egin ondoren aurrera jarraituko du. Hona hemen programa:

```

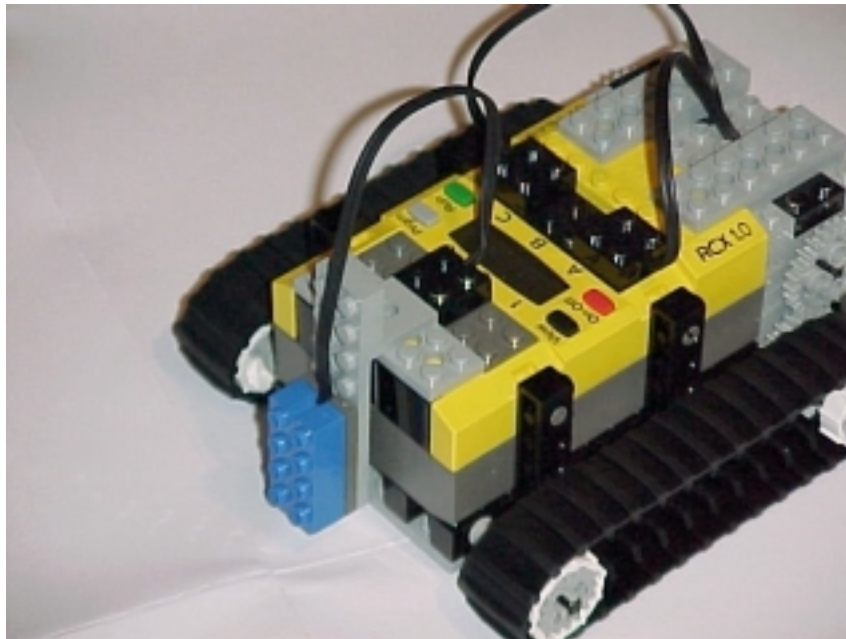
task main()
{
  SetSensor(SENSOR_1,SENSOR_TOUCH);
  OnFwd(OUT_A+OUT_C);
  while (true)
  {
    if (SENSOR_1 == 1)
    {
      OnRev(OUT_A+OUT_C); Wait(30);
      OnFwd(OUT_A); Wait(30);
      OnFwd(OUT_A+OUT_C);
    }
  }
}

```

Aurreko adibidean bezala, lehen urratsa sentsorea konfiguratzea izango da, ondoren, robota martxan jartzeko. Amaigabeko **while** begizta baten bitartez, sentsoreak zerbait ukitzen duen etengabe egiaztatuko dugu. Zerbait ukitzean segundoko hiru hamarrenez atzera egingo du, beste horrenbeste denbora biratzen beteko du berriro aurrera jarraitu aurretik.

## Argi-sentsoreak

Zure MindStorms System-en ukitze-sentsoreekin batera argi-sentsore bat aurkituko duzu. Argi-sentsoreak noranzko zehatz batean iristen zaion argi kantitatea neurtzen du. Sentsoreak argia igortzen du, honela, argi sentsorea noranzko jakin batean jar dezakegu, noranzko horretan objektu batek islatzen duen argia neurtzeko. Hau oso erabilgarria da robot batek lurreko marra bat jarraitu behar duenean, datorren adibidean egin behar dugun moduan. Hasteko, argi-sentsore bat jarriko diogu robotari, aurreko aldearen erdian behera begira. Konekta ezazu 22 sarreran. Robotak irudiko itxura izango du:



RIS<sup>4</sup>-ekin batera datorren zirkuitua ere behar dugu (gainean zirkuitu beltz bat duena). Robotak ziurtatu behar du argi-sentsorea pistaren gainean dagoela. Argi intentsitateak gora egiteak sentsorea pistatik kanpo dagoela esan nahi du, eta honen ondorioz, norabidea zuzendu beharko da. Hemen aurkezten dudun programak arazo honi erantzuten dio, baina bakarrik balio du robota erlojuaren orratzen noranzkoan mugitzen denean.

---

<sup>4</sup> RIS: Robotics Invention System

```

#define MUGA 40

task main()
{
    SetSensor(SENSOR_2, SENSOR_LIGHT);
    OnFwd(OUT_A+OUT_C);
    while (true)
    {
        if (SENSOR_2 > MUGA)
        {
            OnRev(OUT_C);
            until (SENSOR_2 <= MUGA);
            OnFwd(OUT_A+OUT_C);
        }
    }
}

```

Hasieran, programak 2 sarreran konektatutako sentsorea argi-sentsorea dela adierazten du. Ondoren, robota martxan jartzen du eta amaigabeko begizta bati hasiera ematen dio. Argi intentsitatearen balioa 40 balioa gainditzen duenean (honetarako erraz alda daitekeen konstante bat erabiliko dugu, inguruaren argitasunak eragin handia eduki dezakeelako) motore baten biraketa noranzkoa aldatuko dugu, eta robota berriz zirkuitura itzuli arte zain egongo gara.

Programa egikaritzen duzunean, higidura ez da uniformea. Robota hobeto mugitu dadin, **until** aginduaren aurretik `wait(10)` agindua txerta dezakezu. Ohartuko zara programak ez duela funtzionatzen mugimendua erlojuaren orratzen noranzkoaren aurkakoa bada. Robota edozein ibilbide jarraitzeko modukoa izateko korapilatsua den programa bat egin behar da.

## Laburpena

Kapitulu honetan ukitze-sentsorea eta argi-sentsorearen funtzionamenduak ezagutu dituzu. Sentsoreak erabiltzen direnean oso erabilgarria den **until** agindua ere ezagutu duzu.

Fase honetan programa batzuk idaztea komeni zaizu. Zure robotari portaera konplexuak emateko behar diren tresnak eskuratu dituzu. Adibidez, jarri iezazkiozu robotari bi ukitze sentsore, bat aurre aldeko ezkerrean eta bestea eskuinean, robota oztopoak ekiditen mugitu dadin. Egin ezazu robota marra beltz batek mugatzen duen inguru batetik ez ateratzeko behar den programa.

## VI. Atazak eta subrutinak

Orain arte egin ditugun programek ataza bakar bat zuten. Baina NQC-ko programek ataza anitza eduki ditzakete. Gainera, programaren toki ezberdinetan erabilgarriak izango diren kode zatiak subrutina izenekoetan idatz ditzakegu. Ataza eta subrutinak erabiliz programak irakurterazagoak eta trinkoagoak izango dira. Kapitulu honetan eskaintzen dituzten aukera batzuk ikusiko ditugu.

### Atazak

NQC programa bat gehienez 10 atazez osaturik dago. Ataza bakoitzak bere izena izango du, eta horien artean, batek **main** (nagusia) izena izango du. Robotak ataza nagusia egikarrituko du **Run** botoia sakatzean. Gainerako atazak bakarrik ataza nagusiak **start** aginduaren bitartez deitzen dituztenean egikarrituko dira. Hortik aurrera atazak batera egikarrituko dira. Egikaritzen ari den ataza batek beste bat geldi dezake **stop** aginduaren bitartez. Ataza hori aurrerago berriro martxan jartzen bada hasieratik abiatuko da, ez gelditu den tokitik.

Azter dezagun atazen funtzionamendua adibide baten bidez. Jar iezaiozu berriro ukitze-sentsorea robotari. Robotak bere mugimenduz karratuak deskribatu beharko ditu, arestian egin dugun moduan, baina oztoporen batekin talka egiten badu erreakzionatu beharko du. Hau ez da erraza ataza bakar batez egitea, robotak batera bi jarduera bete behar dituelako: gidatu (hau da, behar denean motoreak gelditu eta martxan jarri) eta sentsoreei begiratu. Hau dela eta, hobe da bi ataza gehiago erabiltzea: ataza bat karratuak deskribatzen mugitzeko eta beste bat sentsoreetan gertatzen diren aldaketen aurrean erreakzionatzeko. Hona hemen programa:

```
task main()
{
  SetSensor(SENSOR_1,SENSOR_TOUCH);
  start sentsorea_egiaztatu;
  start karratuan_mugitu;
}

task karratuan_mugitu ()
{
  while (true)
  {
    OnFwd(OUT_A+OUT_C); Wait(100);
    OnRev(OUT_C); Wait(85);
  }
}

task sentsorea_egiaztatu ()
{
  while (true)
  {
    if (SENSOR_1 == 1)
    {
      stop karratuan_mugitu;
      OnRev(OUT_A+OUT_C); Wait(50);
      OnFwd(OUT_A); Wait(85);
      start karratuan_mugitu;
    }
  }
}
```

Programa honetan ataza nagusiak (**main**) bi funtzio betetzen ditu soilik: sentsore mota definitu eta gainerako atazei hasiera eman. Hau egin eta gero **main** ataza bukatzen da. **Karratuan\_mugitu** atazak robota karratutan mugitzen du, **sentsorea\_egiaztatu** ukitze-sentsorea sakatuta dagoen egiaztatzen duen bitartean. Kontakua dagoenean karratuan\_mugitu ataza geldituko du (hau oso garrantzitsua da), eta robotaren kontrola bere gain hartuko du. Robotak atzera egingo du eta biratuko da robotaren kontrola berriz **karratuan\_mugitu** atazari itzultzeko.

Oso garrantzitsua da abiarazitako atazak batera egikaritzen direla gogoan izatea, ezusteko ondorioak ekar ditzakeelako. X. kapituluak arazo hauetan sakontzen du eta irtenbideak eskaintzen ditu.

## Subrutinak

Batzuetan programa baten toki ezberdinetan kode zati berdinak erabili behar dira. Horrelako kasuetan, kode zati hori subrutina moduan idatz daiteke eta izen bat eman. Hortik aurrera kode zati hori bere izenaz deituz atazaren edozein tokitatik deitu ahal izango dugu. NQC-k (eta RCX-k) gehienez 8 subrutina erabil ditzake. Ikus dezagun adibide bat:

```
sub biratu()
{
    OnRev(OUT_C); Wait(340);
    OnFwd(OUT_A+OUT_C);
}

task main()
{
    OnFwd(OUT_A+OUT_C);
    Wait(100);
    biratu ();
    Wait(200);
    biratu ();
    Wait(100);
    biratu ();
    Off(OUT_A+OUT_C);
}
```

Programa honetan biratzeko subrutina bat definitu dugu. Ataza nagusiak (**main**) hiru alditan erabiltzen du subrutina. Subrutina erabiltzeko bere izena idazten da, eta ondoren parentesia. Bere itxura ikusi ditugun NQC-ko agindu askoren itxura du, baina parametrorik ez duenez, parentesi artean ez du ezer.

Hemen arretaz ibili behar da subrutinak bitxi samarrak baitira. Adibidez, subrutina batetik ezin diogu beste subrutina bati deitu. Subrutinak ataza ezberdinetatik dei ditzakegu, baina denak ez dira berri onak nahiko erraz arazoetara eramaten gaituztelako. Subrutina bat ataza ezberdinetan batera egikaritzen denean ezusteko ondorioak dakartza. Bestetik, subrutina bat ataza ezberdinetatik deitu behar badugu ezin dira espresio korapilatsuak erabili firmwareak onartzen ez ditu eta. Beraz, egiten ari zarena oso modu zehatzean ezagutzen ez baduzu, *ez ezazu dei subrutina bat ataza ezberdinetatik*.

## Funtzioak

Subrutinek hainbat arazo sortzen baditu ere, RCX-n behin bakarrik gordetzen dira. Horrek memoria aurreratzen du, eta RCX-k memoria gehiegi ez duenez eskertzekoa da. Baina subrutinak laburrak direnean hobe da funtzioak erabiltzea. Funtzioak ez dira aparte gordetzen baizik eta erabiliko diren toki bakoitzean. Honek memoria gehiago eskatzen du, baina espresio korapilatsuak erabiltzearen ondorio txarrak ekiditen dira. Gainera, funtzio kopuruak ez du mugarik.

Funtzioak subrutinak bezala definitzen eta erabiltzen dira, baina kasu honetan erreserbatutako **void** hitza erabiliko dugu **sub**-en ordean (**void** erabiltzen da beste hainbat programazio hizkuntzetan agertzen delako, adibidez, C-n). Aurreko adibidean funtzioak erabiltzen ditugu programa honela geldituko da:

```

void biratu()
{
    OnRev(OUT_C); Wait(340);
    OnFwd(OUT_A+OUT_C);
}

task main()
{
    OnFwd(OUT_A+OUT_C);
    Wait(100);
    biratu();
    Wait(200);
    biratu ();
    Wait(100);
    biratu ();
    Off(OUT_A+OUT_C);
}

```

Funtzioek beste abantaila bat dute subrutinekin alderatuta: argumentuak eduki ditzakete. Argumentuak funtzioei aldagaiaren baten balioa transferitzeko erabil daitezke. Adibidez, demagun aurreko adibidean biraketa denbora funtzioaren argumentua izatea nahi dugula ondoko adibidean ikus daitekeen moduan:

```

void biratu (int biraketa_denbora)
{
    OnRev(OUT_C); Wait(biraketa_denbora);
    OnFwd(OUT_A+OUT_C);
}

task main()
{
    OnFwd(OUT_A+OUT_C);
    Wait(100);
    biratu (200);
    Wait(200);
    biratu (50);
    Wait(100);
    biratu (300);
    Off(OUT_A+OUT_C);
}

```

Funtzioaren izenaren ondorengo parentesi artean argumentua idatzi dugu. Kasu honetan zenbaki osoa dela (bestelako aukerak ere badaude) eta bere izena biraketa\_denbora dela adierazi diogu. Argumentu bat baino gehiago dagoenean komen bitartez bereiziko ditugu.

## Makroen definizioa

Oraindik ere, kodigo zati laburrei izena emateko beste modu bat dago. NQC-n makroak defini daitezke (kontuz, ez dute zerikusirik RCX Command Center-eko makroekin). Orain aurreko programan hainbat aldaketa egingo dugu eta biraketarako makroak erabiliko ditugu.

```

#define biratu OnRev(OUT_C);Wait(340);OnFwd(OUT_A+OUT_C);

task main()
{
    OnFwd(OUT_A+OUT_C);
    Wait(100);
    biratu;
    Wait(200);
    biratu;
    Wait(100);
    biratu;
    Off(OUT_A+OUT_C);
}

```

**#define** instrukzioaren ondoren biratu hitza eta ordezkatzeko dituen aginduak idatzi ditugu. Hortik aurrera, biratu idazten duzun bakoitzean dagokion testuarekin ordezkatzeko da. Ohartuko zinen testu osoa lerro bakar batean idatzita dagoela. Hala ere, makro bati dagokion kode zatia lerro ezberdinetan idazteko moduak badaude, baina, ez da gomendagarria.

**define** instrukzioak potentzia handikoak dira: argumentuak erabil ditzakete. Adibidez, biraketa denbora instrukzioaren argumentu bezala jar dezakegu. Hemen lau makro definitzen dituen adibide bat eskaintzen da: bat aurrera egiteko, beste bat atzera egiteko, hiruargarrena ezkerrean biratzeko eta laugarrena eskuinera biratzeko. Bakoitzak bi argumentu erabiltzen ditu: abiadura eta denbora.

```
#define eskuinera(s,t)  SetPower(OUT_A+OUT_C,s);OnFwd(OUT_A);OnRev(OUT_C);Wait(t);
#define ezkerre(s,t)   SetPower(OUT_A+OUT_C,s);OnRev(OUT_A);OnFwd(OUT_C);Wait(t);
#define aurrera(s,t)   SetPower(OUT_A+OUT_C,s);OnFwd(OUT_A+OUT_C);Wait(t);
#define atzera(s,t)    SetPower(OUT_A+OUT_C,s);OnRev(OUT_A+OUT_C);Wait(t);

task main()
{
    aurrera (3,200);
    ezkerre(7,85);
    aurrera (7,100);
    atzera(7,200);
    aurrera (7,100);
    eskuinera(7,85);
    aurrera(3,200);
    Off(OUT_A+OUT_C);
}
```

Makroak definitzea oso lagungarria da. Kodea trinkoagoa eta irakurgarriagoa bilakatzen du. Gainera, kodean moldaketak egitea errazten du, adibidez, motoreen konexioak aldatzen dituzunean.

## Laburpena

Kapitulu honetan ataza, subrutina funtzio eta makroak nola erabiltzen diren ikasi duzu. Atazak gehienetan batera egikaritzen dira eta batera egin behar diren jardueraz arduratzen dira. Subrutinak berdinak diren kode zati luzeak ataza baten toki ezberdinetan erabili behar direnean erabiltzen dira. Funtzioak kode zati berdinak ataza ezberdinetako toki ezberdinetan agertzen direnean erabilgarriak dira, baina memoria gehiago behar dute. Bukatzeko, makroak toki ezberdinetan agertzen diren kode zati txikiak ordezkatzeko erabiltzen dira. Gainera parametroak eduki ahal izateak erabilgarriagoak bilakatzen ditu.

Hona iritsi bazara, roboten portaera konplexuak emateko behar dena ikasi duzu. Eskuliburu honetako gainerako kapituluetan jasotzen dena aplikazio batzuetan bakarrik garrantzitsuak izango diren beste kontu batzuk jasotzen dira.

## VII. Soinuak eta musika

RCX-k soinuak igortzeaz gain musika konposizio errazak jo ditzakeen bozgorailu bat du. Aukera hau oso interesgarria da zerbait gertatzean RCX-k horren berri ematea nahi dugunean. Gainera, dibertigarria izan daiteke robotak mugitzen den bitartean, musika jotzea.

### Soinu aukera

RCX-k sei soinu ezberdin egin ditzake. 0 eta 5 bitarteko zenbakiez ezagutzen diren soinuak hauek dira:

- 0 Teklako klika
- 1 Txistua
- 2 Beherakako maiztasuneko ekorketa
- 3 Gorakako maiztasuneko ekorketa
- 4 “Buhhh” Errore soinua
- 5 Goraka azkar doan ekorketa Barrido de rápido crecimiento

Soinu hauek igortzeko `PlaySound()` agindua erabiltzen da. Hona hemen denak jotzen dituen programa bat:

```
task main()  
{  
    PlaySound(0); Wait(100);  
    PlaySound(1); Wait(100);  
    PlaySound(2); Wait(100);  
    PlaySound(3); Wait(100);  
    PlaySound(4); Wait(100);  
    PlaySound(5); Wait(100);  
}
```

Baina, zer dela eta idatzi ditugu **wait** aginduak? Soinua sortzen duen agindua egikaritzean ez da zain gelditzen soinua bukatu arte, berehala hurrengo agindua egikaritzen baitu. RCX-k buffer txiki bat du soinu batzuk gordetzeko, baina berehala betetzen da eta soinuak galtzen dira. Hau ez da larria soinuak igortzen direnean, baina garrantzi handia du musikaren kasuan, jarraian ikusiko dugun bezala.

`PlaySound()` argumentuak derrigorrez konstante bat izan behar du, ez du aldagairik onartzen.

### Musika jotzen

Musika jotzeko NQC-k `PlayTone()` agindua eskaintzen du. Bi argumentu du: lehenengoa maiztasuna eta bigarrena iraupena (segundo baten ehunenetan **wait** aginduan bezala). Hona hemen maiztasun erabilgarrienak:

Nota	1	2	3	4	5	6	7	8
G#	52	104	208	415	831	1661	3322	
G	49	98	196	392	784	1568	3136	
F#	46	92	185	370	740	1480	2960	
F	44	87	175	349	698	1397	2794	
E	41	82	165	330	659	1319	2637	
D#	39	78	156	311	622	1245	2489	
D	37	73	147	294	587	1175	2349	
C#	35	69	139	277	554	1109	2217	
C	33	65	131	262	523	1047	2093	4186
B	31	62	123	247	494	988	1976	3951
A#	29	58	117	233	466	932	1865	3729
A	28	55	110	220	440	880	1760	3520

Soinuen kasuan ikusi dugun moduan, RCXk ez du itxaroten soinua bukatu arte. Beraz, jarraian nota asko jartzen badituzu hobe da **wait** aginduak tartekatzea. Ikus dezagun adibide batean:

```

task main()
{
    PlayTone(262,40); Wait(50);
    PlayTone(294,40); Wait(50);
    PlayTone(330,40); Wait(50);
    PlayTone(294,40); Wait(50);
    PlayTone(262,160); Wait(200);
}

```

Hau egiteko modurik errazena RCX Command Center-ek eskaintzen duen RCX Piano (Tools menuan) da.

Robot bat mugitzen den bitartean musika jotzea nahi baduzu hobe da horretarako ataza bat erabiltzea. Jarraian ematen den adibidean RCX-k aurrera eta atzera egiten du musika etengabe jotzen duen bitartean.

```

task musika()
{
    while (true)
    {
        PlayTone(262,40); Wait(50);
        PlayTone(294,40); Wait(50);
        PlayTone(330,40); Wait(50);
        PlayTone(294,40); Wait(50);
    }
}

task main()
{
    start musika;
    while(true)
    {
        OnFwd(OUT_A+OUT_C); Wait(300);
        OnRev(OUT_A+OUT_C); Wait(300);
    }
}

```

## Laburpena

Kapitulu honetan robotak soinuak igortzeko eta musika jotzeko nola programatu behar duzun ikasi duzu. Musikarako aparteko ataza bat nola erabili ere ikasi duzu.

## VIII. Motoreetan sakontzen

Motoreak zehaztasun handiagoz kontrolatzeko beste agindu batzuk daude. Kapitulu honetan aztertuko ditugu.

### Emeki geldituz

`Off()` agindua erabiltzen duzunean motorea berehala gelditzen da balaztak erabiliz. NQC-n motorea gelditzeko beste era bat dago, gozoagoa balaztak erabili gabe. Horretarako `Float()` agindua erabiltzen du. Batzuetan, honela egitea egokiagoa da robotaren atazarako. Jarraian eskaintzen den adibidean hasieran robota balaztak erabiliz geldituko da, eta ondoren erabili gabe. Kasu honetan bien arteko aldea ez da handia, baina beste robot batzuekin handiagoa izan daiteke.

```
task main()
{
    OnFwd(OUT_A+OUT_C);
    Wait(200);
    Off(OUT_A+OUT_C);
    Wait(100);
    OnFwd(OUT_A+OUT_C);
    Wait(200);
    Float(OUT_A+OUT_C);
}
```

### Agindu aurreratuak

`OnFwd()` aginduak bi zeregin betetzen ditu: motorea martxan jarri eta aurrerako noranzkoa finkatu. `OnRev()` aginduak ere bi zeregin betetzen ditu: motorea martxan jarri eta atzerako noranzkoa finkatu. NQC bi zeregin hauek agindu berezietan egin dezake. Bietako bat bakarrik aldatu nahi duzunean hobea da berezitako aginduak erabiltzea: RCX-n memoria gutxiago erabiltzen du, azkarragoa da eta mugimendu gozoagoak eragiten ditu. Berezirik erabil daitezkeen aginduak `SetDirection()`, noranzkoa finkatzen duena (`OUT_FWD`, `OUT_REV` edo `OUT_TOGGLE`, uneko noranzkoa alderantzikatzen duena), eta `SetOutput()`, modua finkatzen duena (`OUT_ON`, `OUT_OFF` edo `OUT_FLOAT`). Ondoko programa errazari esker robotak aurrera egingo du, atzera eta bukatzeko berriz aurrera.

```
task main()
{
    SetPower(OUT_A+OUT_C, 7);
    SetDirection(OUT_A+OUT_C, OUT_FWD);
    SetOutput(OUT_A+OUT_C, OUT_ON);
    Wait(200);
    SetDirection(OUT_A+OUT_C, OUT_REV);
    Wait(200);
    SetDirection(OUT_A+OUT_C, OUT_TOGGLE);
    Wait(200);
    SetOutput(OUT_A+OUT_C, OUT_FLOAT);
}
```

Programa guztietan hasieran motoreen noranzkoa aurrerakoa da eta potentzia 7 mailakoa, beraz, aurreko programako lehenengo bi lerroak ez dira beharrezkoak.

Badaude beste agindu batzuk gehiago motoreak kontrolatzeko ikusitakoen laburdurak direnak. Jarraian zerrenda osoa eskaintzen da:

<code>On('motoreak')</code>	Motoreak konektatzen ditu.
<code>Off('motoreak')</code>	Motoreak deskonektatzen ditu.
<code>Float('motoreak')</code>	Motoreak balaztak erabili gabe gelditzen ditu,
<code>Fwd('motoreak')</code>	Motoreak aurrerako noranzkoan (baina ez ditu mugiarazten)
<code>Rev('motoreak')</code>	Motoreak atzerako noranzkoan (baina ez ditu mugiarazten)
<code>Toggle('motoreak')</code>	Motoreen noranzkoa alderantzikatzen du
<code>OnFwd('motoreak')</code>	Motoreak aurrerako noranzkoan martxan jartzen ditu
<code>OnRev('motoreak')</code>	Motoreak atzerako noranzkoan martxan jartzen ditu
<code>OnFor('motoreak', 'tik')</code>	Motorea denbora zehatz batez (tik-etan emanda) martxan jartzen du

`SetOutput('motoreak', 'modua')` Irteerako modua definitzen du (`OUT_ON`, `OUT_OFF` edo `OUT_FLOAT`)  
`SetDirection('motoreak', 'dir')` Irteerako noranzkoa definitzen du (`OUT_FWD`, `OUT_REV` edo `OUT_TOGGLE`)  
`SetPower('motoreak', 'potentzia')` Irteerako potentzia finkatzen du (0-7)

## Motorearen abiaduraren aldaketa

Beharbada ohartu zara motoreen abiaduraren aldaketak ez duela eragin handirik. Arrazoa pareta aldatzen dela da, eta ez abiadura. Ondorioak motoreak karga handia duenean soilik antzemango dituzu, nahiz eta kasu horretan ere 2 eta 7 arteko aldea txikia izan. Emaitza hobea lortu nahi badituzu, trikimailu bat erabili beharko duzu: motorea behin eta berriro azkar konektatu eta deskonektatu. Jarraian hori egiteko programa erraz bat eskaintzen da. Ataza batek, *motorea\_martxan* izenekoak, motoreak gidatzen ditu eta *abiadura* aldagaia etengabe egiaztatzen du uneko abiadura zein den jakiteko. Balio positiboa aurreranzko mugimenduari dagokio eta negatiboa atzeranzkoari. Motoreei noranzko egokia eman ondoren denbora tarte batez zain gelditzen da, berriz gelditu aurretik. Ataza nagusiaren funtzio bakarra abiadurak eta itxaronaldiak finkatzea da.

```
int abiadura, _ abiadura;

task motorea_martxan ()
{
    while (true)
    {
        _abiadura = abiadura;
        if (_abiadura > 0) {OnFwd(OUT_A+OUT_C);}
        if (_abiadura < 0) {OnRev(OUT_A+OUT_C); _abiadura = -_abiadura;}
        Wait(_abiadura);
        Off(OUT_A+OUT_C);
    }
}

task main()
{
    speed = 0;
    start motorea_martxan;
    abiadura = 1;    Wait(200);
    abiadura = -10; Wait(200);
    abiadura = 5;   Wait(200);
    abiadura = -2;  Wait(200);
    stop motorea_martxan;
    Off(OUT_A+OUT_C);
}
```

Programa hau eraginkorragoa bilaka daiteke, adibidez, biraketak onartuz eta `Off()` aginduaren ondoren itxaronaldi bat sartuz. Egin itzazu proba batzuk zure kontura.

## Laburpena

Kapitulu honetan motoreentzat dauden agindu gehigarriak ikasi dituzu: `Float()`, motorea gozoki gelditzen duena, `SetDirection()`, noranzkoa finkatzen duena (`OUT_FWD`, `OUT_REV` edo `OUT_TOGGLE` noranzkoa alderantzikatzeke) eta `SetOutput()`, modua finkatzen duena (`OUT_ON`, `OUT_OFF` edo `OUT_FLOAT`). Motoreekin zerikusia duten aginduen zerrenda osoa ikusi duzu, eta motoreen abiadura kontrolatzeko trikimailu bat ezagutu duzu.

## IX. Sentsoreetan sakontzen

V kapituluaren sentsoreak erabiltzeko oinarriko aginduak aztertu ditugu. Baina sentsoreek askoz aukera gehiago eskaintzen dituzte. Kapitulu honetan sentsore-mota eta sentsore-moduaren artean dauden diferentziak aztertuko ditugu. Gainera, errotazio-sentsoreak nola erabili (RISekin saltzen ez den sentsorea, baina aparte eskuratu daitekeena<sup>5</sup>), hiru sentsore baino gehiago batera erabiltzeko modua eta hurbiltasun sentsorea nola egin ikusiko dugu.

### Sentsore mota eta moduak

V kapituluaren ikusi dugun `SetSensor()` aginduak bi zeregin betetzen ditu: sentsore-mota eta erabiltzen den modua definitzea. Sentsore-modua eta sentsore-mota bakoitza bere aldetik finkatuz sentsorearen portaera doitasun handiagoz kontrola dezakezu. Honela egitea oso baliagarria da hainbat aplikazioetan.

Sentsore-mota finkatzeko `SetSensorType()` agindua erabiltzen da. Lau sentsore-mota ezberdin dago: `SENSOR_TYPE_TOUCH` (ukitze-sentsorea), `SENSOR_TYPE_LIGHT` (argi-sentsorea), `SENSOR_TYPE_TEMPERATURE` (tenperatura-sentsorea<sup>6</sup>) eta `SENSOR_TYPE_ROTATION` (errotazio-sentsorea). Sentsore-mota finkatzea garrantzi handikoa da, batez ere, sentsorea elikatu behar den adierazteko (argi-sentsorearekin gertatzen den moduan). Ez dago sentsoreren bat jatorrizko motan ez finkatzea zer edo zertarako interesgarria den.

Sentsore-modua finkatzeko `SetSensorMode()` agindua erabiltzen da. Zortzi modu daude aukeran, eta horien artean garrantzitsuenak `SENSOR_MODE_RAW` da. Modu honetan, sentsorea txekiatzen denean eskuratzen den balioa 0 eta 1023 artekoa da. Hau da sentsoreak ematen duen balioa prozesatu gabe. Zenbaki honen esanahia sentsorearen arabera da. Adibidez, ukitze-sentsorearen kasuan, sakatuta ez dagoenean irakurketaren balioa 1023tik gertukoa da. Erabat sakatuta dagoenean 50 inguru eta erdi sakatuta dagoenean 500 eta 1000 artekoa. Beraz, ukitze-sentsore bat RAW moduan dagoenean erdi sakatuta dagoen antzeman daiteke. Sentsorea argi-sentsorea denean balioak 300 (argia asko) eta 800 (argi gutxi) artekoa izaten dira. Honela lortzen dugun irakurketa `SetSensor()` agindua erabiltzen dugunean baino askoz zehatzagoa da.

Bigarren modua `SENSOR_MODE_BOOL` da. Modu honetan eskuratzen den balioa 0 ala 1 da. Prozesatu gabeko balioa 550-etik gorakoa bada balioa 0 izango da, eta gainerakoetan 1. `SENSOR_MODE_BOOL` ukitze-sentsoreen modu lehenetsia da. `SENSOR_MODE_CELSIUS` eta `SENSOR_MODE_FAHRENHEIT` moduak bakarrik tenperatura-sentsorearekin dira erabilgarri, eta tenperatura aukeratutako unitateetan ematen dute. `SENSOR_MODE_PERCENT` prozesatu gabeko balioak 0 eta 100 bitarteko balio bihurtzen ditu. Prozesatu gabe 400 baino txikiagoa den balio orori %100 dagokio, eta hori baino handiagoa bada, pixkanaka zeroraino jaisten da. `SENSOR_MODE_PERCENT` argi-sentsorearen modu lehenetsia da. `SENSOR_MODE_ROTATION` errotazio-sentsorearekin bakarrik omen da erabilgarri (aurrerago ikusiko dugu).

Intereseko beste bi modu dago: `SENSOR_MODE_EDGE` eta `SENSOR_MODE_PULSE`. Modu hauetan iragaiteak zenbatzen dira, hau da, prozesatu gabeko balio txiki batetik balio handi batera iragatea edo alderantzizkoa. Adibidez, ukitze sentsore bat sakatzean balio altu batetik balio baxu batera iragaite bat gertatzen da. Askatzean aurkako noranzkoan beste iragaite bat gertatzen da. Sentsore modua `SENSOR_MODE_PULSE` denean, balio altu batetik balio baxu baterako iragaiteak zenbatzen dira, beraz, ukitze sentsorea sakatzen eta askatzen den bakoitzean iragaite bat zenbatuko da. Baina, sentsore modua `SENSOR_MODE_EDGE` denean, bi iragaiteak zenbatuko dira, hau da, sentsorea sakatzen eta askatzen denan bi unitate gehituko dira. Ukitze sentsore bat zenbat alditan sakatzen den zenbatzeko erabil daiteke edo argi-sentsore batekin argi bat zenbat alditan pizten eta itzaltzen den. Gertaerak zenbatzen direnean, kontadorea zeron jartzeko aukera izan behar dugu. Horretarako `ClearSensor()` agindua erabiltzen da. Aipatutako sentsorearen (edo sentsoreen) kontadorea zeron jartzen du.

Ikus dezagun adibide bat. Hurrengo programa ukitze-sentsore bat erabiltzen du robot bat gidatzeko. Konekta ezazu ukitze-sentsore bat 1 sarreran kable luze baten bitartez. Sentsorea modu azkarrean bitan sakatzen baduzu aurrera egingo du, baina behin bakarrik sakatuz geldituko da.

---

<sup>5</sup>Itzulpen hau egin den uenean “Ultimate Accesory Set”-ean errotazio-sentsorea saltzen da urrutiko aginte batekin (Itzultzailearen oharra)

<sup>6</sup> Sentsore hau ez da RISen aurkitzen, baina eros daiteke aparte.

```

task main()
{
    SetSensorType(SENSOR_1,SENSOR_TYPE_TOUCH);
    SetSensorMode(SENSOR_1,SENSOR_MODE_PULSE);
    while(true)
    {
        ClearSensor(SENSOR_1);
        until (SENSOR_1 >0);
        Wait(100);
        if (SENSOR_1 == 1) {Off(OUT_A+OUT_C);}
        if (SENSOR_1 == 2) {OnFwd(OUT_A+OUT_C);}
    }
}

```

Beharbada ohartu zara sentsore-mota sentsore-moduaren aurretik definitzen dela. Hau garrantzi handikoa da, sentsore-mota aldatzeak sentsore-moduan ondorioak dituelako.

## Errotazio-sentsorea

Errotazio sentsorea oso erabilgarria da, baina ez da RIS estandarrekin saltzen. Aparte erosi daitezke. Errotazio sentsoreak zulo bat du ardatz bat sartzeko. Sentsoreak arbatzaren biraketa neurtzen du. Ardatzak itzuli bat ematen duenean 16 pauso zenbatzen ditu (edo -16 biraketaren noranzkoa aurkakoa bada). Errotazio-sentsorea robotak doitasunez kontrolatutako mugimenduak egiteko erabil daitezke: nahi duzun mugimendua lor dezakezu. Behar duzun kontrolak itzuliko 16 pauso baino gehiago eskatzen baditu, engranajeen bitartez azkarrago mugitzen den ardatz bati konekta dezakezu eta pausoak zenbatu<sup>7</sup>.

Ohiko aplikazio bat bi errotazio-sentsoreak bi motorez mugitzen den robot baten gurpiletan konektatzea da. Mugimendu zuzen bat lortzeko bi aldetako gurpilak abiadura berdinean biratu behar dira. Zorritzarez, gehienetan bi motore ez dira abiadura berdinean biratzen. Errotazio-sentsoreei esker zein gurpil doan azkarrago antzeman dezakezu. Motore hori denbora baterako gelditu dezakezu (`Float()` agindua erabiliz) bi sentsoreek balio berdina eman arte. Hori da hurrengo programak erakusten duena. Zure robotean aldaketa batzuk egin behar dituzu errotazio-sentsoreak gurpiletan konektatzeko. Konekta itzazu 1 eta 3 sarreretan.

```

task main()
{
    SetSensor(SENSOR_1,SENSOR_ROTATION); ClearSensor(SENSOR_1);
    SetSensor(SENSOR_3,SENSOR_ROTATION); ClearSensor(SENSOR_3);
    while (true)
    {
        if (SENSOR_1 < SENSOR_3)
            {OnFwd(OUT_A); Float(OUT_C);}
        else if (SENSOR_1 > SENSOR_3)
            {OnFwd(OUT_C); Float(OUT_A);}
        else
            {OnFwd(OUT_A+OUT_C);}
    }
}

```

Programa hasieran sentsoreak errotazio-sentsoreak direla adierazten da, eta, beren balioak zeron jartzen dira. Ondoren amaigabeko begizta bati hasiera ematen zaio. Begizta honetan bi sentsoreen irakurketak berdinak diren egiaztatzen da. Hala bada, robotak aurrera jarraituko du. Ostera, bat bestea baino handiago bada alde horretako motorea geldituko da bien irakurketa berdindu arte.

Hau oso programa erraza da. Robotak distantzia zehatz bat egiteko behar diren aldaketak egin ditzakezu, edo biraketak zehaztasunez egin ditzan.

<sup>7</sup> Sentsoreak fidagarritasunez neurtzeko onartzen duen abiadurarik handiena 500 rpm-koa da (itzultzailearen oharra).

## Nola jarri sentsore bat baino gehiago sarrera batean

RCX-k hiru sarrera baino ez ditu, beraz, hiru sentsore bakarrik jar daitezke. Robot landuagoak egin nahi badituzu hau ez da nahikoa (batez ere sentsore gehiago erosi badituzu). Zorionez, trikimailuren bat erabiliz bi sentsore (edo gehiago) konekta daitezke sarrera batean.

Bi ukitze sentsore sarrera batean konektatzea oso erraza da. Bietako bat sakatzen denean (edo biak batera) irakurketaren balioa 1 izango da, bestela 0. Ez duzu jakingo zein izan den, baina kasu askotan ez da beharrezkoa. Adibidez, robot bati ukitze-sentsore bat aurrean eta beste bat atzean jartzen dizkiozunean, motorearen noranzkoari begiratuta zein izan den jakingo duzu. Baina, sentsoreak RAW moduan konfiguratzen badituzu (ikus 27. orrialdea) askoz informazio gehiago eskuratuko duzu. Zoriterik baduzu sakatuta daudenean ematen duten irakurketa ez da berdina izango bi sentsoreetan. Hala bada, ez duzu biak bereizteko arazorik izango, eta biak batera sakatzen hori ere antzeman ahal izango duzu irakurketa askoz txikiagoa izango delako (30 inguru).

Beste aukera bat ukitze-sentsore bat eta argi-sentsore bat sarrera berean konektatzea da. Sentsore motak argi-sentsoreari dagokiona izan behar du, bestela argi-sentsoreak ez du funtzionatuko. Sentsore modua RAW izango da, honela, ukitze-sentsorea sakatzean eskuratuko den irakurketa 100 baino txikiagoa izango da. Ukitze-sentsorea sakatuta ez dagoenean argi-sentsorearen irakurketa eskuratuko da (beti 100 baino altuagoa). Jarraian eskaintzen den programa ideia honetan oinarritzen da: argi-sentsore bat lurrari begira eta ukitze-sentsore bat aurreko kolpe-leungailuan izango ditu, biak 1 sarreran konektatuta. Robotak era aleatorioan mugituko da argizaturiko inguru batean. Argi-sentsoreak lurtean marra beltz bat aurkitzen duenean (RAW balioa >750) pixka bat atzera egingo du. Ukitze-sentsoreak zerbait ukitzean (RAW balioa < 100) ere atzera egingo du. Hona hemen programa:

```
int ttt,tt2;

task mugitu_aleat()
{
    while (true)
    {
        ttt = Random(50) + 40;
        tt2 = Random(1);
        if (tt2 > 0)
            { OnRev(OUT_A); OnFwd(OUT_C); Wait(ttt); }
        else
            { OnRev(OUT_C); OnFwd(OUT_A); Wait(ttt); }
        ttt = Random(150) + 50;
        OnFwd(OUT_A+OUT_C); Wait(ttt);
    }
}

task main()
{
    start mugitu_aleat;
    SetSensorType(SENSOR_1,SENSOR_TYPE_LIGHT);
    SetSensorMode(SENSOR_1,SENSOR_MODE_RAW);
    while (true)
    {
        if ((SENSOR_1 < 100) || (SENSOR_1 > 750))
        {
            stop mugitu_aleat;
            OnRev(OUT_A+OUT_C); Wait(30);
            start mugitu_aleat;
        }
    }
}
```

Uste dut programaren nondik norakoa argi dagoela. Bi ataza ditu: nagusia eta mugitu\_aleat. mugitu\_aleat ataza robotak era aleatorioan mugitzen du. Ataza nagusiak sentsoreak konfiguratuta ondoren gertaeraren baten zain gelditzen da. Sentsoreak oso irakurketa baxua (zerbait ukitu) edo altuegia (inguru zuritik kanpo) ematen badu robotak pixka bat atzera egingo du berriz mugimendu aleatorioan berriz hasi aurretik.

Bi argi-sentsore sarrera bakar batean konekta baditzaiegu ere, pixka bat nahasia eta erabiltzeko zaila da, nahiz eta Raw balioa eta bi sentsoreek irakurritako argi intentsitateen artean nolabaiteko harremana egon. Argi edo ukitze-sentsore bat errotazio edo tenperatura-sentsoreekin batera konektatzea ez dirudi oso erabilgarria denik.

## Nola egin hurbiltasun-sentsore bat

Ukitze-sentsoreei esker zure robotak erreakzionatu dezake talka egiten duenean. Baina, ikusgarriago da talka egin aurretik erreakzionatzea. Horretarako oztopotik gertu dagoela jakin beharko luke baina ez dago horretarako sentsore propiorik<sup>8</sup>. Hala ere, trikimailu bat erabiliz lor dezakegu. RCX-k infragorrien ataka bat du ordenagailuarekin edo beste robotekin komunikatzeko (XI kapituluan roboten arteko komunikazioak aztertuko ditugu) eta RIS-ekin batera datozen argi-sentsoreak oso sentikorak dira argi infragorriarekiko. Honetan guztian oinarrituta hurbiltasun-sentsore bat egin dezakegu. Ataza batek mezua igorriko ditu argi infragorriaz eta beste batek objektuek islatzen duten argi infragorriaren gorabeherak neurtuko ditu. Gorabeherak handiak direnean objektu batetik gertu gaudela jakingo dugu.

Idea hau gauzatzeko jar ezazu argi-sentsore bat robotaren infragorrien atakaren gainean aurrera begira. Honela, islatutako argi infragorria neurtuko du soilik. Konekta ezazu 2 sarreran. Irakurketen gorabeherak doitasun handiagoz antzemateko RAW modua erabiliko dugu argi-sentsorearekin. Jarraian eskaintzen den programari esker robotak aurrera egingo du objektu batera gerturatu arte, orduan, 90°ko biraketa egingo du eskuin aldera.

```
int azken_irak; // aurreko irakurketa gordetzeko

task seinalea_igorri()
{
    while(true)
        {SendMessage(0); Wait(10);}
}

task seinalea_txekeatu()
{
    while(true)
    {
        azken_irak = SENSOR_2;
        if(SENSOR_2 > azken_irak + 200)
            {OnRev(OUT_C); Wait(85); OnFwd(OUT_A+OUT_C);}
    }
}

task main()
{
    SetSensorType(SENSOR_2, SENSOR_TYPE_LIGHT);
    SetSensorMode(SENSOR_2, SENSOR_MODE_RAW);
    OnFwd(OUT_A+OUT_C);
    start seinalea_igorri;
    start seinalea_txekeatu;
}
```

seinalea\_igorri atazak 10 IR seinale igortzen ditu segundoko `SendMessage(0)` agindua erabiliz. seinalea\_txekeatu atazak argi-sentsorearen balioa etengabe gordetzen du. Gainera, irakurketak 200 unitateko gorakada bat izan duen egiaztatzen du, hori baita gorabehera handi baten adierazlea. Hala bada, robotak 90°ko biraketari ekingo dio. 200eko balioa arbitrarioa da. Txikiagoa bada, robotak urrutiagotik antzemango ditu oztopoa, baina hori materiala eta inguruan dagoen argi intentsitatearen araberakoa da. Balio egokia hautatzeko esperimendazioa da biderik onena.

Teknika honen eragozpen bat norabide bakar batean lan egitea da. Beharbada, robotaren alboetan ukitze-sentsoreak jarri beharko dituzu talkak ekiditeko. Teknika hau oso erabilgarria da labirintotan mugitu behar diren robotentzat. Beste eragozpen bat ordenagailutik robotarekin komunikatzeko ezintasuna da, robotak igortzen dituen mezuekin interferentziak ekar ditzakeelako (telebistaren urrutiko agintearen funtzionamenduan ere arazoak sor ditzake).

---

<sup>8</sup>:LegoStuff-ek hurbiltasun sentsore bat eskaintzen du (itzultzailearen oharra).

## Laburpena

Kapitulu honetan sentsoreei buruz gauza gehiago ikusi dugu. Sentsore-mota eta modua bakoitza bere aldetik nola finkatu ikasi duzu, baita informazio gehigarria eskuratzeko bidea ere. Orain badakizu nola erabiltzen den errotazio-sentsorea eta nola jar daitekeen sentsore bat baino gehiago RCXko sarrera bakar batean. Bukatzeko, hurbiltasun-sentsore bat egiteko trikimailu bat ikusi dugu. Horretarako, robotaren infragorrien ataka argi-sentsore batekin konbinatu dugu. Trikimailu hauek oso interesgarriak dira robot sofistikatuagoak egiteko, sentsoreen erabilpenak berebiziko garrantzia baitu.

## X. Ataza paraleloak

Arestian esan dugun moduan, NQC-n ataza batera egikaritzen dira, edo paraleloan, hizkuntza arruntean esaten den moduan. Hau oso erabilgarria da: ataza batetik sentsoreei begira gauden bitartean, beste batek robota mugitu dezake eta hirugarren batek musika jo dezake. Baina paraleloan egikaritzen diren atazek arazoak sor ditzakete, ataza batek beste bat oztopa dezakeelako.

### Programa akastun bat

Azter dezagun ondoko programa. Hemen ataza batek robota gidatzen du karratutan mugitu dadin (lehen egin dugun bezala) eta bigarren ataza batek ukitze-sentsoreari begiratzen dio. Sentsorea sakatzean pixka bat egiten du atzera eta 90°ko biraketa egiten du.

```
task main()
{
    SetSensor(SENSOR_1,SENSOR_TOUCH);
    start sntsoreari_begira;
    while (true)
    {
        OnFwd(OUT_A+OUT_C); Wait(100);
        OnRev(OUT_C); Wait(85);
    }
}

task sentsoreari_begira()
{
    while (true)
    {
        if (SENSOR_1 == 1)
        {
            OnRev(OUT_A+OUT_C);
            Wait(50);
            OnFwd(OUT_A);
            Wait(85);
            OnFwd(OUT_C);
        }
    }
}
```

Beharbada, programa biribila iruditzen zaizu. Baina, baliteke egikaritzen duzunean ezusteko portaerak aurkitzea. Egin ezazu ondoko proba: robota biratzen denean oztopo bat uki dezala. Atzeraka hasiko da, baina berehala, berriz egingo du aurrera oztopoarekin talka eginez. Honen arrazoia atazen arteko interferentziak dira. Ondoko hau gertatzen da: robota eskuin aldera biratzen ari denean, hau da, lehen ataza bigarren wait aginduan dagoenean, oztopoarekin talka egiten du eta atzeraka hasten da, baina une horretan ataza nagusia prest dago eta robota aurrera darama berriz oztopoarekin talka egin arte. Bigarren ataza ez da gertatutako talkaz ohartuko lotan dagoelako (wait agindua). Hau ez da espero genuen portaera. Bigarren ataza lo dagoen artean lehena oraindik martxan dago, eta bere ekintzek bigarren atazaren ekintzetan interferentziak sortzen dituzte.

### Atazen gelditzea eta berrabiaraztea

Arazo hau ekiditeko modu bat une guztietan robota ataza bakar batek gidatuko duela ziurtatzea da. Orain VI. kapituluaren ikusi dugun programan aldaketa batzuk egingo ditugu.

```

task main()
{
    SetSensor(SENSOR_1,SENSOR_TOUCH);
    start sentsoreei_begira;
    start mugitu_karratuan;
}

task mugitu_karratuan ()
{
    while (true)
    {
        OnFwd(OUT_A+OUT_C); Wait(100);
        OnRev(OUT_C); Wait(85);
    }
}

task sentsoreei_begira ()
{
    while (true)
    {
        if (SENSOR_1 == 1)
        {
            stop mugitu_karratuan;
            OnRev(OUT_A+OUT_C); Wait(50);
            OnFwd(OUT_A); Wait(85);
            start mugitu_karratuan;
        }
    }
}

```

Programa honetan, sentsoreei\_begira ataza bakarrik mugituko du robota mugitu\_karratuan ataza gelditu ondoren. Honela, ataza honek ez du trabarik sortuko oztopotik urruntzeko mugimenduan. Behin atzerakako prozedura bukatuta, berriro abiatzen da mugitu\_karratuan ataza.

Aurreko arazoa konpontzeko irtenbide ona bada ere, arazo bat sortzen da. mugitu\_karratuan berriro abiaraztean, hasiera-hasieratik egiten du. Hau, gure ataza xumean ez da garrantzitsua, baina beste batzuetan ez da gogoko portaera. Nahiago genuke ataza edozein tokitan gelditu ondoren berriro toki beretik jarraitzea. Tamalez, ez da gauza erraza.

## Semaforoaren erabilpena

Atazen artean interferentziarik ez sortzeko bide arrunt bat zein ataza kontrolatzen duen motorea adierazten duen aldagai bat erabiltzea da. Beste atazek ez dute baimenik izango motoreak erabiltzeko lehenengoak prest dagoela adierazi arte, semaforoaren bitartez. Sarritan holako aldagai bati semaforo izena emango zaio. sem aldagaia semaforo bat izango da. Motorea ataza baten kontrolpean ez dagoenean aldagaiaren balioa 0 izango da, eta ataza batek motoreekin zerbait egin nahi duenean atazak ondoko aginduak egikaritzeko ditu:

```

until (sem == 0);
sem = 1;
// motoreekin zerbait egin
sem = 0;

```

Hasieran itxarongo dugu motoreak libre egon arte. Une horretan, motoreen kontrola eskuratuko dugu sem aldagaiari 1 balioa emanez. Hortik aurrera motoreak kontrolatuko ditugu. Bete beharreko lana bukatzean berriz esleituko diogu 0 balioa sem aldagaiari. Aurreko programan semaforoaren erabilpena inplementatu dugu. Ukitze-sentsoreak zerbait ukitzen duenean semaforoa aktibatzen da eta atzerakako prozedura egikaritzen da. Prozedura hau egikaritzen den bitartean mugitu\_karratuan ataza zain gelditzen da. Atzerakako mugimendua bukatzen den unean, semaforoaren balioa 0-ra itzultzen da eta mugitu\_karratuan atazak martxan jarraitzen du.

```

int sem;

task main()
{
    sem = 0;
    start mugu_karratua;
    SetSensor(SENSOR_1,SENSOR_TOUCH);
    while (true)
    {
        if (SENSOR_1 == 1)
        {
            until (sem == 0); sem = 1;
            OnRev(OUT_A+OUT_C); Wait(50);
            OnFwd(OUT_A); Wait(85);
            sem = 0;
        }
    }
}

task mugu_karratua ()
{
    while (true)
    {
        until (sem == 0); sem = 1;
        OnFwd(OUT_A+OUT_C);
        sem = 0;
        Wait(100);
        until (sem == 0); sem = 1;
        OnRev(OUT_C);
        sem = 0;
        Wait(85);
    }
}

```

Pentsa dezakezu ez dela beharrezkoa mugu\_karratua atazan semaforoari 1 balioa ematea eta ondoren berriz 0 baliora itzularaztea, baina komenigarria da. Arrazoiak OnFwd() instrukzioa bi aginduz osaturik dagoela da (ikus IX kapituluan), eta honela egiten ez bada, beste atazak agindu sekuentzia hori eten dezake

Semaforoak oso erabilgarriak dira, eta paraleloan egikaritzen diren atazak dituen programa korapilotsu bat idazten denean gehienetan beharrezkoak izaten dira (badago huts egiteko beste aukera bat ere, ea asmatzen duzun zergatik).

## Laburpena

Kapitulu honetan atazak paraleloan erabiltzen dituzunean sor daitezkeen arazoak ezagutu dituzu. Kontu handiz ibili behar izaten da ezusteko ondorioekin. Ezusteko asko atazak paraleloan erabiltzearen ondorioak dira.. Arazo hauei irtenbidea emateko bi bide aztertu dugu: lehenengoak atazak gelditzen eta abiatzen ditu une bakoitzean ataza kritiko bakar bat egikaritzen dela ziurtatzeko; eta bigarrenak atazan egikaritzea kontrolatzeko semaforoak erabiltzen ditu. Honela une bakoitzean ataza bakar bat egikaritzen dela bermatuko da.

## XI. Roboten arteko komunikazioak

RCX bat baino gehiago baduzu kapitulu hau zuzentzat da. Robotak haien artean komunika daitezke infragorrien atakaren bitartez. Honela, robotak elkarlanean (edo elkarren aurka borrokan) jar ditzakezu. Robot handiagoak egiteko aukera izango duzu, bi RCXekin sei motore eta sei sentsore (edo gehiago IX kapituluko trikimailuei esker) kontrola baititzakezu.

Roboten arteko komunikazioa, orokorrean, honela funtzionatzen du: robot batek `SendMessage()` agindua erabil dezake mezu bat (0 eta 255 arteko balioa) igortzeko infragorrien ataka erabiliz. Gainerako robotek mezua jaso eta gordetzen dute. Robotaren programak jasotako azken mezua edukia `Message()` agindua erabiliz eskura dezake, eta balio horretan oinarrituz robotak ekintza bat edo beste egingo du.

### Aginduen igorpena

Normalean, robot bat baino gehiago dagoenean bat buruzagia izaten da. *Nagusia* izena eman diezaiokegu. Gainerakoak *esklaboak* izango dira. Robot nagusiak aginduak igortzen ditu eta esklaboek egikaritzen dituzte. Batzuetan esklaboek informazioa itzultzen dio, adibidez, sentsore baten irakurketa. Beraz, bi programa idatzi behar dituzu: bat nagusiarentzat eta beste bat (batzuk) esklaboarentzat (esklaboentzat). Hemendik aurrera esklabo bat dugula suposatuko dugu. Has gaitzen programa simple batekin. Esklaboak hiru agindu ezberdin egikari ahal izango ditu: aurrera, atzera eta gelditu. Programa oinarria begizta labur bat da: uneko mezuari `ClearMessage()` aginduaren bitartez 0 balioa eman ondoren zain gelditzen da balio hori berriz aldatu arte. Balioaren arabera hiru aginduetako bat egikarituko du.

```
task main() // ESKLABUA
{
    while (true)
    {
        ClearMessage();
        until (Message() != 0);
        if (Message() == 1) {OnFwd(OUT_A+OUT_C);}
        if (Message() == 2) {OnRev(OUT_A+OUT_C);}
        if (Message() == 3) {Off(OUT_A+OUT_C);}
    }
}
```

Patroiaren programa askoz sinpleagoa da: aginduei dagozkien mezuak igortzen ditu eta pixka bat itxaroten du. Ondorengo programan nagusiak esklaboari ondoko hau egiteko eskatzen dio: aurrera egin, bi segundoz itxaron, atzera beste bi segundoz eta bukatzeko gelditu.

```
task main() // NAGUSIA
{
    SendMessage(1); Wait(200);
    SendMessage(2); Wait(200);
    SendMessage(3);
}
```

Bi programa hauek idatzi ondoren robotetara transferitu behar dituzu. Programa bakoitza robot batera transferitu behar da. Ziurta ezazu beste robota itzalita dagoela transferentzia egiten den bitartean (ikus aurrerago gogoan eduki beharreko gomendioak). Orain, konekta itzazu bi robotak eta abiaraz itzazu programak: lehen esklaboarena eta ondoren nagusiarena.

Esklabo bat baino gehiago baduzu programak txandaka transferitu behar dituzu (batera ez, ikus aurrerago). Orain esklabo guztiek ekintza bera egingo dute batera.

Robotak elkarrekin komunikatzeko protokolo bat definitu dugu: 1 aurrera izango da, 2 atzera eta 3 gelditu. Garrantzi handikoa da protokoloak kontu handiz definitzea, batez ere, komunikazio ugari dagoenean. Adibidez, esklabo gehiago dagoenean bi zenbaki igorri ditzakezu (bien artean tarte labur bat utziaz): lehen zenbakia esklaboaren identifikadorea bigarrena unez uneko agindua. Esklaboak zenbakia egiaztatzen du, eta ekintza egikaritzen du bakarrik berea bada. Honela egiteko robot bakoitzak zenbaki ezberdina eduki behar du, beraz, esklabo bakoitzaren programa pixka bat ezberdina izango da, adibidez, konstante baten balioa aldatuz.

## Nagusiaren izendapena

Aurreko atalean ikusi dugun moduan, robot ugari dagoenean robot bakoitzak bere programa eduki behar du. Askoz errazagoa litzateke robot guztiak programa berdina balute. Baina kasu horretan, galdera hau litzateke: Zein da nagusia? Erantzuna erraza da: utz diezaiegun robotei erabakitzen. Baina, nola egin daiteke hori? Ideia simple samarra da. Robot bakoitzak denbora kopuru aleatorioa itxarongo du mezu bat igorri aurretik. Mezua igortzen duen lehena nagusia izango da. Estrategia honek kale egin dezake bi robotek mezua batera igortzen badute, baina nekeez gertatuko da (estrategia korapilatsuagoak diseina ditzakezu holako ezustekoak ekiditeko). Hona hemen estrategia hau erabiltzen duen programa:

```
task main()
{
    ClearMessage();
    Wait(200);
    Wait(Random(400)); // ziurta ezazu denak martxan daudela
    if (Message() > 0) // 0 eta 4 segundoen artean zain
    { // lehenengoa den egiaztatzen du
        start esklaboa;
    }
    else
    {
        SendMessage(1); // Ni naiz nagusia
        Wait(400); // besteek dakitela ziurtatu
        start nagusia;
    }
}

task nagusia ()
{
    SendMessage(1); Wait(200);
    SendMessage(2); Wait(200);
    SendMessage(3);
}

task esklaboa()
{
    while (true)
    {
        ClearMessage();
        until (Message() != 0);
        if (Message() == 1) {OnFwd(OUT_A+OUT_C);}
        if (Message() == 2) {OnRev(OUT_A+OUT_C);}
        if (Message() == 3) {Off(OUT_A+OUT_C);}
    }
}
```

Transferi ezazu programa hau robot guztietara (txandaka, denak batera ez, ikus aurrerago). Jar itzazu robotak martxan batera eta beha ezazu zer gertatzen den. Baten batek agintea hartuko du, eta gainerakoek aginduak beteko dituzte. Kasuren batean nagusia ez aukeratzea gerta daiteke. Arestian esan dugun moduan, holako egoerak ekiditeko protokolo landuagoak behar dira.

## Kontuan hartzekoa

Kontu handiz ibili behar da robot batzuk erabiltzen direnean. Bi arazo sor daiteke: bi robotek (edo robot batek eta ordenagiluek) informazioa batera igortzen badute, galdu daiteke; eta bigarren arazoa ordenagailuak programa robot batzuetara batera bidaltzen duenean gertatzen da.

Azter dezagun bigarren arazoa. Programa ordenagailura transferitzen duzunean, robotak programa (programaren zatiak) behar bezala jaso duen jakinarazten dio ordenagailuari. Ordenagailua erantzunez beste programa zatiak bidaltzen ditu, edo gaizki jasotakoak berriz bidaltzen ditu. Bi robot piztuta daudenean, biak hasiko dira ordenagailuari programa behar bezala jaso duten jakinarazten. Ordenagailuak ez du hori ulertuko (berak ez baitaki bi robot daudela), beraz, gauzak okertuko dira eta programa hondatuko da. Programak ez du bere ataza beteko. *Ziurta ezazu programak transferitu bitartean robot bakarra dagoela piztuta.*

Robotak edozein unetan bidal dezake mezu bat, eta horrek azaroak ekar ditzake. Bi mezu ia batera igortzen badira gal daitezke. Gainera, robotak ezin ditu mezuak igorri eta jaso batera. Hau ez da arazoa robot bakar batek mezuak igortzen dituenean (nagusi bakarra dagoenean) baina hala ez bada arazo handia ekar dezake. Adibidez, pentsa ezazu idatzi duzun programan esklaboak talka egiten duenean mezu bat igortzen diola nagusiari, nagusiak erabaki bat har dezan. Baina une horretan bertan nagusiak mezu bat bidaltzen badu, gal daiteke. Irtenbidea komunikazio arazoak gainditzeko gai den protokolo egoki batean datza. Adibidez, nagusiak mezu bat igortzen duenean esklaboaren erantzuna jaso behar du. Erantzuna behar bezain azkar jasotzen ez badu, berriro igorriko du agindua. Horrelako zerbait egingo luke ondoko kode zatiak:

```
do
{
    SendMessage(1);
    ClearMessage();
    Wait(10);
}
while (Message() != 255);
```

Hemen 255 erabili da erantzuna emateko.

Batzuetan, bi robot baino gehiago dagoenean, gertuen dagoen robotak bakarrik mezua jasotzea nahi izaten dugu. Hau nagusiaren programari `SetTxPower(TX_POWER_LO)` agindua txertatuz lor dezakegu. Honela, igorriko duen seinalea oso ahula izango da, eta bakarrik, gertu dagoen robotak “entzun” ahal izango du. Hau interesgarria da bereziki bi RCX baino gehiagorekin robot handi bat egiten denean. Berriro irismen luzean konfiguratzeko erabil ezazu `SetTxPower(TX_POWER_HI)` agindua.

## Laburpena

Kapitulu honetan roboten arteko komunikazioen alderdi batzuk aztertu ditugu. Komunikazioetan mezuak igortzeko, txekeatzeko eta ezabatzeko aginduak erabiltzen dira. Komunikazioen funtzionamendua protokolo batean definitu behar direla ikusi dugu. Honelako protokoloek berebiziko garrantzia dute ordenagialuen arteko komunikazioetan. Roboten arteko komunikazioetan hainbat muga daudela ikusi dugu, eta horren ondorioz protokolo onak definitzea garrantzi handikoa dela.

## XII. Agindu gehiago

NQC-k beste agindu batzuk ditu. Kapitulu honetan hiru agindu mota ikusiko dugu: tenporizadoreen erabilpena, display-a kontrolatzeko aginduak eta RCX-k datuak erregistratzeko duen gaitasuna.

### Tenporizadoreak

RCX-k barruko lau tenporizadore ditu. Tenporizadore hauek tik-tak egiten dute segundo baten hamarrenetan.

Tenporizadoreen izenak 0 eta 3 bitarteko zenbakiak dira. Tenporizadoreen balioa berrasieratzeko

`ClearTimer()` agindua erabiliko dugu, eta balio bat esleitzeko `Timer()` agindua. Jarraian tenporizadore baten erabilera bat aurkezten da. Programa honek gidatuta robota 20 segundoz modu aleatorioan mugituko da.

```
task main()
{
    ClearTimer(0);
    do
    {
        OnFwd(OUT_A+OUT_C);
        Wait(Random(100));
        OnRev(OUT_C);
        Wait(Random(100));
    }
    while (Timer(0)<200);
    Off(OUT_A+OUT_C);
}
```

Programa hau IV kapituluaren ataza berdina betetzeko egindako programarekin konpara dezakezu. Dударik gabe, tenporizadorea erabiltzen duena sinpleagoa da.

Tenporizadoreak `wait()` agindua ordezkatzeko erabil daitezke. Iraupen zehatzeko itxaronaldia egiteko tenporizadorea berrasiera dezakezu, eta balio zehatz batera iritsi arte itxaron. Baina zain zauden bitartean, beste gertaera baten aurrean erreakziona dezake (adibidez, sentsoreen aldaketa baten aurrean). Ondorengo programa honen adibide erraz bat da. Robota 10 segundoz aurrera egingo du, edo sentsoreak zerbait ukitu arte.

```
task main()
{
    SetSensor(SENSOR_1,SENSOR_TOUCH);
    ClearTimer(3);
    OnFwd(OUT_A+OUT_C);
    until ((SENSOR_1 == 1) || (Timer(3) >100));
    Off(OUT_A+OUT_C);
}
```

Ez ahaztu tenporizadoreek segundo hamarrenetan lan egiten dutela, Wait aginduak, adibidez,segundo ehunenetan egiten duen artean.

### Pantaila

RCX-ren pantaila bi era ezberdinetara kontrola daiteke. Lehenengoan, sistemaren erlojua, sentsoreetako bat, edo motoreetako bat erakusten du. Hau, RCX-ko botoi beltza (View) erabiltzea bezala da. Pantaila mota aukeratzeko `SelectDisplay()` agindua erabiltzen da. Ondoko programak zazpi aukerak aurkezten ditu, bata bestearen atzetik.

```

task main()
{
  SelectDisplay(DISPLAY_SENSOR_1); Wait(100); // 1 sarrera
  SelectDisplay(DISPLAY_SENSOR_2); Wait(100); // 2 sarrera
  SelectDisplay(DISPLAY_SENSOR_3); Wait(100); // 3 sarrera
  SelectDisplay(DISPLAY_OUT_A);    Wait(100); // A irteera
  SelectDisplay(DISPLAY_OUT_B);    Wait(100); // B irteera
  SelectDisplay(DISPLAY_OUT_C);    Wait(100); // C irteera
  SelectDisplay(DISPLAY_WATCH);    Wait(100); // Sistemaren erlojua
}

```

Ezin da `SelectDisplay(SENSOR_1)` erabili.

Pantaila kontrolatzeko beste modu bat sistemaren erlojuaren balioan datza. `SetWatch()` agindua erabiliz diagnostiko informazioak erakutsi ahal izango ditu. Jarraian eskaintzen den programa laburrak erabiltzen du.

```

task main()
{
  SetWatch(1,1); Wait(100);
  SetWatch(2,4); Wait(100);
  SetWatch(3,9); Wait(100);
  SetWatch(4,16); Wait(100);
  SetWatch(5,25); Wait(100);
}

```

`SetWatch()` aginduaren argumentuak derrigorrez konstanteak izango dira.

## Datuen erregistroa

RCX-k aldagaien balioak, sentsoreen irakurketak eta tenporizadoreen balioak erregistra ditzake datalog izeneko memoria inguru batean. RCX-k ezin ditu erabili datalog-en gordetako balioak, baina ordenagailutik irakur ditzakegu. Hau oso erabilgarria da RCX-n gertatzen dena egiaztatzeko. RCX Command Center-ek datalog-en edukia ikusteko tresnak eskaintzen ditu.

Datuak erregistratzeko hiru urrats hauek jarraitu behar dira: Hasieran NQC-k datalog-en tamaina definituko du `CreateDatalog()` agindua erabiliz. Honek aurreko dalalog-en edukia ezabatuko du. Bigarren urratsa, balio bat beste baten atzean erregistratzea izango da (RCX-ren pantailan disko baten sektoreak banan-banan agertuko dira, eta diskoa osatzen denean datalog-a betea dagoela jakingo dugu). Datalog-aren bukaera iristen bada ez da ezer gertatzen, balio berriak ez dira erregistratuko. Hirugarren urratsa datalog-a ordenagailura transferitzea da. Horretarako RCX Command Center-en **Tools** (tresnak) menuko **Datalog** aukera hautatuko dugu. Ondoren, saka ezazu **Upload Datalog** botoia eta balioak agertuko dira. Bertan ikus ditzakezu, edo bestela, fitxategi batean gorde. Badago ezaugarri honetan oinarrituz RCX-rekin eskaner bat egin duena.

Hona hemen, argi-sensore bat duen robot batekin egindako adibidea. Robotak 10 segundoz aurrera egingo du, eta argi-sensorearen irakurketa bost aldiz segundoko erregistratuko ditu dalalog-ean.

```

task main()
{
  SetSensor(SENSOR_2, SENSOR_LIGHT);
  OnFwd(OUT_A+OUT_C);
  CreateDatalog(50);
  repeat (50)
  {
    AddToDatalog(SENSOR_2);
    Wait(20);
  }
  Off(OUT_A+OUT_C);
}

```

### XIII. NQC-ren erreferentzia azkarra

Kapitulu honetan NQC-ko elementu guztien sintaxia aurkituko duzu (aginduak, konstanteak...). Gehienak aurreko kapituluetan aztertu direnez, hemen deskribapen laburra baino ez da ematen.

#### Instrukzioak

Instrukzioa	Deskribapena
<b>while</b> ( <i>bald</i> ) <i>inst.mult.</i>	Instrukzio multzoa baldintza egiazkoa den bitartean egikaritzen du
<b>do</b> <i>inst.mult.</i> <b>while</b> ( <i>bald</i> )	Instrukzio multzoa baldintza egiazkoa den bitartean egikaritzen du
<b>until</b> ( <i>bald</i> ) <i>inst.mult.</i>	Instrukzio multzoa baldintza egiazkoa izan arte egikaritzen du (zero aldiz edo gehiagotan)
<b>break</b>	while/do/until baten instrukzio multzotik atera
<b>continue</b>	while/do/until baten instrukzio multzoaren hurrengo iterazioa egikaritu gabe utzi
<b>repeat</b> ( <i>espresioa</i> ) <i>inst.mult</i>	Instrukzio multzoa espresioak adierazitako aldi kopurua errepikatu.
<b>if</b> ( <i>bald</i> ) <i>inst1</i> <b>if</b> ( <i>bald</i> ) <i>inst1</i> <b>else</b> <i>inst2</i>	<i>Inst1</i> baldintza benetakoa denean egikarituko du, eta <i>inst2</i> (baldin badago) baldintza betetzen ez denean
<b>start</b> <i>ataza_izena</i>	Aipatutako ataza hasieratzen du
<b>stop</b> <i>ataza_izena</i>	Aipatutako ataza gelditzen du
<b>function</b> ( <i>args</i> )	Funtzioari dei egiten dio behar diren argumentuak erabiliz
<b>var</b> = <i>espresioa</i>	Espresio bat balioztatzen du eta aldagai bati esleitzen dio
<b>var</b> += <i>espresioa</i>	Espresio bat balioztatzen du eta aldagai bati gehitzen dio
<b>var</b> -= <i>espresioa</i>	Espresio bat balioztatzen du eta aldagai bati kentzen dio
<b>var</b> *= <i>espresioa</i>	Espresio bat balioztatzen du eta aldagaia bider espresioa egiten du
<b>var</b> /= <i>espresioa</i>	Espresio bat balioztatzen du eta aldagaia zati espresioa egiten du
<b>var</b>  = <i>espresioa</i>	Espresio bat balioztatzen du eta aldagaiarekin OR bitez biteko konparaketa egiten du, azken emaitza aldagaiari esleituz
<b>var</b> &= <i>espresioa</i>	Espresio bat balioztatzen du eta aldagaiarekin AND bitez biteko konparaketa egiten du, azken emaitza aldagaiari esleituz
<b>return</b>	Funtzioaren kontrola deitu dionari itzultzen dio
<i>espresioa</i>	Balioztatu beharreko espresioa

#### Baldintzak

Baldintzak kontrol egituretan erabiltzen dira erabakiak hartzeko. Gehienetan, baldintza espresioen arteko konparaketa baten bitartez definitzen da.

Baldintza	Esanahia
<b>true</b>	Beti egiazkoa
<b>false</b>	Beti faltsua
<i>espr1</i> == <i>espr2</i>	Espresioak berdinak diren egiaztatzen du
<i>espr1</i> != <i>espr2</i>	Espresioak ezberdinak diren egiaztatzen du
<i>espr1</i> < <i>espr2</i>	Espresio bat bestea baino txikiagoa den egiaztatzen du
<i>espr1</i> <= <i>espr2</i>	Espresio bat bestea baino txikiagoa edo berdina den egiaztatzen du
<i>espr1</i> > <i>espr2</i>	Espresio bat bestea baino handiagoa den egiaztatzen du
<i>espr1</i> >= <i>espr2</i>	Espresio bat bestea baino handiagoa edo berdina den egiaztatzen du
! baldintza	Baldintzaren ezeztapen logikoa
<i>bald1</i> && <i>bald2</i>	Bi baldintzen arteko AND logikoa (baldin eta soilik baldin bi baldintzak egiakoak badira)
<i>bald 1</i>    <i>bald 2</i>	Bi baldintzen arteko OR logikoa (baldin eta soilik baldin gutxienez baldintza bat egiazkoa bada)

## Espresioak

Espressoetan mota ezberdinetako balioak erabil daitezke, besteak beste, konstanteak, aldagaiak eta sentsoreen irakurketak. `SensorValue(0)`, `SensorValue(1)`, eta `SensorValue(2)`-ren ordez `SENSOR_1`, `SENSOR_2`, eta `SENSOR_3` makroak erabil daitezke.

Balioa	Deskribapena
<code>zenbakia</code>	Balio konstante bat (adib. "123")
<code>aldagaia</code>	Aldagai baten izena (adib. "x")
<code>Timer(n)</code>	n tenporizadorearen balioa (n 0 eta 3 artekoa)
<code>Random(n)</code>	0 eta n arteko zenbaki aleatorioa
<code>SensorValue(n)</code>	n sentsorearen balioa (n 0 eta 2 artekoa)
<code>Watch()</code>	Sistema erlojuaren balioa
<code>Message()</code>	Jasotako azken IR mezuaren balioa

Balioak operadore ezberdinak erabiliz konbina daitezke. Operadore batzuk bakarrik espresio konstanteak ebaluatzeko erabil daitezke, eta derrigorrezkoa da espresioaren elementu guztiak konstanteak izatea. Ondoko operadoreen zerrenda lehentasunaren arabera ematen dira (lehentasun handitik txikira).

Operadorea	Deskribapena	Elkartzea	Mugak	Adibidea
<code>abs()</code>	Balio absolutua	-	Ez	<code>abs(x)</code>
<code>sign()</code>	Eragigaiaren zeinua	-	Ez	<code>sign(x)</code>
<code>++</code>	Gehikuntza	ezkerra	Aldagaiak bakarrik	<code>x++</code> edo <code>++x</code>
<code>--</code>	Gutxipena	ezkerra	Aldagaiak bakarrik	<code>x--</code> edo <code>--x</code>
<code>-</code>	Unary minus	eskuina	Ez	<code>-x</code>
<code>~</code>	Bit mailako ukapena	eskuina	Konstanteak bakarrik	<code>~123</code>
<code>*</code>	Biderketa	ezkerra	Ez	<code>x * y</code>
<code>/</code>	Zatiketa	ezkerra	Konstanteak bakarrik	<code>x / y</code>
<code>%</code>	Modulua	ezkerra	Ez	<code>123 % 4</code>
<code>+</code>	Batuketa	ezkerra	Ez	<code>x + y</code>
<code>-</code>	Kenketa	ezkerra	Ez	<code>x - y</code>
<code>&lt;&lt;</code>	Biten desplazamendua ezker aldera	ezkerra	Konstanteak bakarrik	<code>123 &lt;&lt; 4</code>
<code>&gt;&gt;</code>	Biten desplazamendua eskuin aldera	ezkerra	Konstanteak bakarrik	<code>123 &gt;&gt; 4</code>
<code>&amp;</code>	AND bit mailan	ezkerra	Ez	<code>x &amp; y</code>
<code>^</code>	XOR bit mailan	ezkerra	Konstanteak bakarrik	<code>123 ^ 4</code>
<code> </code>	OR bit mailan	ezkerra	Ez	<code>x   y</code>
<code>&amp;&amp;</code>	AND logikoa	ezkerra	Konstanteak bakarrik	<code>123 &amp;&amp; 4</code>
<code>  </code>	OR logikoa	ezkerra	Konstanteak bakarrik	<code>123    4</code>

## RCX funtzioak

Funtzio gehienek argumentuek espresio konstanteak izan behar dute (zenbakiak edo espresio konstanteen arteko eragiketak). Salbuespenak sentsoreak argumentu moduan erabiltzen dituzten funtzioak dira. Sentsoreen kasuan argumentuak sentsore baten izena izan beharko luke: `SENSOR_1`, `SENSOR_2`, edo `SENSOR_3`. Kasu batzuetan konstanteek aurrez definitutako izena dute (adibidez, `SENSOR_TOUCH`).

Funtzioa	Deskribapena	Adibidea
<code>SetSensor(sentsorea, konfig)</code>	Sentsore bat konfiguratu du	<code>SetSensor(SENSOR_1, SENSOR_TOUCH)</code>
<code>SetSensorMode(sentsorea, modua)</code>	Sentsore-moduaren konfigurazioa	<code>SetSensor(SENSOR_2, SENSOR_MODE_PERCENT)</code>
<code>SetSensorType(sentsorea, mota)</code>	Sentsore-motaren konfigurazioa	<code>SetSensor(SENSOR_2, SENSOR_TYPE_LIGHT)</code>
<code>ClearSensor(sentsorea)</code>	Sentsore baten balioa ezabatzen du	<code>ClearSensor(SENSOR_3)</code>

<b>Funtzioa</b>	<b>Deskribapena</b>	<b>Adibidea</b>
<code>On(irteerak)</code>	Irteera bat, edo gehiago, konektatzen du.	<code>On(OUT_A + OUT_B)</code>
<code>Off(irteerak)</code>	Irteera bat, edo gehiago, deskonektatzen du.	<code>Off(OUT_C)</code>
<code>Float(irteerak)</code>	Irteerak balaztarik gabe gelditzen ditu.	<code>Float(OUT_B)</code>
<code>Fwd(irteerak)</code>	Irteeretan aurrerako noranzkoa finkatzen du.	<code>Fwd(OUT_A)</code>
<code>Rev(irteerak)</code>	Irteeretan atzerako noranzkoa finkatzen du.	<code>Rev(OUT_B)</code>
<code>Toggle(irteerak)</code>	Irteeretako noranzkoa alderantzikatzen du.	<code>Toggle(OUT_C)</code>
<code>OnFwd(irteerak)</code>	Martxan jartzen du aurrera.	<code>OnFwd(OUT_A)</code>
<code>OnRev(irteerak)</code>	Martxan jartzen du atzera.	<code>OnRev(OUT_B)</code>
<code>OnFor(irteerak, denbora)</code>	Martxan jartzen du adierazitako segundo ehunenatarako. Denbora espresio bat izan daiteke.	<code>OnFor(OUT_A, 200)</code>
<code>SetOutput(irteerak, modua)</code>	Irteerako modua finkatzen du.	<code>SetOutput(OUT_A, OUT_ON)</code>
<code>SetDirection(irteerak, nora)</code>	Irteerako noranzkoa finkatzen du	<code>SetDirection(OUT_A, OUT_FWD)</code>
<code>SetPower(irteerak, potentzia)</code>	Irteerako potentzia maila finkatzen du (0-7). Potentzia espresio bat izan daiteke.	<code>SetPower(OUT_A, 6)</code>
<code>Wait(denbora)</code>	Adierazitako denboraz itxaroten du. Denbora segundo ehunenetan ematen da eta espresio bat izan daiteke.	<code>Wait(x)</code>
<code>PlaySound(soinua)</code>	Aipaturako soinua egiten du (0-5).	<code>PlaySound(SOUND_CLICK)</code>
<code>PlayTone(maizt, iraupena)</code>	Adierazitako maiztasuneko toinoa jotzen du adierazitako denboraz (segundo hamarrenetan).	<code>PlayTone(440, 5)</code>
<code>ClearTimer(temporizadorea)</code>	Temporizadorea (0-3) hasieratzen du (0 balioa)	<code>ClearTimer(0)</code>
<code>StopAllTasks()</code>	Egikaritzen ari diren ataza guztiak gelditzen ditu.	<code>StopAllTasks()</code>
<code>SelectDisplay(modua)</code>	Pantailaren 7 moduetako bat hautatzen du: 0 – sistemako erlojua; 1..3 – sentsorearen balioa; 4..6 – irteeretako konfigurazioak. Modua espresio bat izan daiteke.	<code>SelectDisplay(1)</code>
<code>SendMessage(mezua)</code>	IR mezu bat igortzen du (1-255). Mezua espresio bat izan daiteke.	<code>SendMessage(x)</code>
<code>ClearMessage()</code>	IR mezuen bufferra ezabatzen du	<code>ClearMessage()</code>
<code>CreateDatalog(tamaina)</code>	Adierazitako tamainako datuen erregistroa sortzen du.	<code>CreateDatalog(100)</code>
<code>AddToDatalog(balioa)</code>	Datuen erregistroari datu bat gehitzen dio. Balioa espresio bat izan daiteke.	<code>AddToDatalog(Timer(0))</code>
<code>SetWatch(orduak, minutuak)</code>	Sistemako erlojuaren balioa finkatzen du.	<code>SetWatch(1, 30)</code>
<code>SetTxPower(hi_lo)</code>	Infragorrien transmisorea goiko edo beheko potentzian finkatzen du.	<code>SetTxPower(TX_POWER_LO)</code>

## RCX konstanteak

RCX-ko funtzioen balioak konstanteen bidez eman daitezke kode irakurgarriagoa bilakatzeko. Ahal den bitartean, hobe da konstante bat erabiltzea balio gordin bat baino.

<a href="#">SetSensor()</a> instrukziorako konfigurazioak	<a href="#">SENSOR_TOUCH</a> , <a href="#">SENSOR_LIGHT</a> , <a href="#">SENSOR_ROTATION</a> , <a href="#">SENSOR_CELSIUS</a> , <a href="#">SENSOR_FAHRENHEIT</a> , <a href="#">SENSOR_PULSE</a> , <a href="#">SENSOR_EDGE</a>
<a href="#">SetSensorMode()</a> -ren moduak	<a href="#">SENSOR_MODE_RAW</a> , <a href="#">SENSOR_MODE_BOOL</a> , <a href="#">SENSOR_MODE_EDGE</a> , <a href="#">SENSOR_MODE_PULSE</a> , <a href="#">SENSOR_MODE_PERCENT</a> , <a href="#">SENSOR_MODE_CELSIUS</a> , <a href="#">SENSOR_MODE_FAHRENHEIT</a> , <a href="#">SENSOR_MODE_ROTATION</a>
<a href="#">SetSensorType()</a> -ren sentso-re-motak	<a href="#">SENSOR_TYPE_TOUCH</a> , <a href="#">SENSOR_TYPE_TEMPERATURE</a> , <a href="#">SENSOR_TYPE_LIGHT</a> , <a href="#">SENSOR_TYPE_ROTATION</a>
<a href="#">Irteera izenak (On(), Off())...</a> aginduekin erabilgarriak	<a href="#">OUT_A</a> , <a href="#">OUT_B</a> , <a href="#">OUT_C</a>
<a href="#">SetOutput()</a> -ren moduak	<a href="#">OUT_ON</a> , <a href="#">OUT_OFF</a> , <a href="#">OUT_FLOAT</a>
<a href="#">SetDirection()</a> -ren noranzkoak	<a href="#">OUT_FWD</a> , <a href="#">OUT_REV</a> , <a href="#">OUT_TOGGLE</a>
<a href="#">SetPower()</a> -ren irteerako potentziak	<a href="#">OUT_LOW</a> , <a href="#">OUT_HALF</a> , <a href="#">OUT_FULL</a>
<a href="#">PlaySound()</a> -ren soinuak	<a href="#">SOUND_CLICK</a> , <a href="#">SOUND_DOUBLE_BEEP</a> , <a href="#">SOUND_DOWN</a> , <a href="#">SOUND_UP</a> , <a href="#">SOUND_LOW_BEEP</a> , <a href="#">SOUND_FAST_UP</a>
<a href="#">SelectDisplay()</a> -ren moduak	<a href="#">DISPLAY_WATCH</a> , <a href="#">DISPLAY_SENSOR_1</a> , <a href="#">DISPLAY_SENSOR_2</a> , <a href="#">DISPLAY_SENSOR_3</a> , <a href="#">DISPLAY_OUT_A</a> , <a href="#">DISPLAY_OUT_B</a> , <a href="#">DISPLAY_OUT_C</a>
<a href="#">SetTxPower()</a> -ren potentzia mailak	<a href="#">TX_POWER_LO</a> , <a href="#">TX_POWER_HI</a>

## Erreserbatutako hitzak

NQC-k hitz batzuk erreserbatzen ditu berak erabiltzeko. Horrelako hitzaren bat funtzioak, ataza edo aldagaiak izendatzeko erabiltzeak erroreak sortuko ditu. Erreserbatutako hitzak hauek dira: **\_\_sensor**, **abs**, **asm**, **break**, **const**, **continue**, **do**, **else**, **false**, **if**, **inline**, **int**, **repeat**, **return**, **sign**, **start**, **stop**, **sub**, **task**, **true**, **void**, **while**.

## XIV. Azken oharra

Eskuliburu jarraitu bitartean zure kontura beste programa batzuk garatu badituzu, NQCn adituzat jo dezakezu zure burua. Oraindik ez baduzu egin, hau da zure kontura esperimentatzen hasteko unea. Diseinuan eta programazioan sormena erabiliz robot mirengarriak egin daitezke.

Eskuliburu honek ez ditu RCX Command Center-eko aukera guztiak erakusten. Bere dokumentazioko atal batzuk irakurtzea gomendatzen dizut. NQC oraindik garatze fasean dago, beraz, bertsio berriek funtzionalitate berriak eskaini ditzakete. Eskuliburu honetan ez dira programazioaren kontzeptu asko jorratu: besteak beste, roboten ikaste portaerak eta adimen artifizialarekin zerikusia dutenak.

LEGO robota PCtik zuzenean ere gida daiteke. Horretarako Visual Basic, Java edo Delphi bezalako programazio hizkuntza batean idatzi beharko duzu zure programa. Programa horiek RCX-n gordetako NQC programa batekin batera ere lan egin dezakete. Konbinaketa hauek oso eraginkorrak dira. Robota modu horretan programatzen hasi nahi baduzu, hobe duzu hasteko spirit-en erreferentzia teknikoa LEGO MindStorms-en web gunean eskuratzea.

<http://www.legomindstorms.com/>

Internet informazio gehigarria eskuratzeko iturri aparta da. Beste abiapuntu interesgarri batzuk nire webgunearen loturak atalean dituzu:

<http://www.cs.uu.nl/people/markov/lego/>

eta LUGNET, LEGO® Users Group Network-n (ez-ofiziala):

<http://www.lugnet.com/>

lugnet.com-en, lugnet.robotics eta lugnet.robotics.rcx.nqc foroetan informazio asko eskura dezakezu.

## **XV. Itzultzailearen oharra**

Nire izena Koldo Olaskoaga da (Donostia). Hezkuntzan robotikaren erabilpenaz proiektu bat egiten ari naiz, eta beste hainbat gauzen artean, dokumentu honen itzulpena egin dut.

Zure ikastetxean erabiltzen baduzu, eskertuko nizuke horren berri ematea.

Nire posta helbidea [robotek@donospat.net](mailto:robotek@donospat.net)

Proiektuaren web orria: <http://www.donospat.net/2000/>